Digital Storybooks to Teach Children Computer Science and Address Common Misconceptions

Thomas S. Deverin

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Sally Hamouda, Chair

Cliff A. Shaffer

Mohammed F. Farghally

April 18, 2025

Blacksburg, Virginia

Keywords: K-8 Education, Computer Science Education, Computational Thinking, Misconceptions

Digital Storybooks to Teach Children Computer Science and Address Common Misconceptions

Thomas S. Deverin

(ABSTRACT)

CodeKids is a web-based platform developed to teach computational concepts to elementary and middle school-aged children. The platform hosts many interactive digital books that teach various computational concepts from bits and bytes to artificial intelligence using themes that children can easily relate to. This study focuses on books created to address common misconceptions students form when learning foundational computer science concepts including variables, conditions, and loops. The primary objectives of this research are to evaluate the students' perceptions of the digital storybook format and compare the misconception rates of students who used the storybooks to those from previous studies. A study was conducted at a local middle school with 6th and 7th-grade students who used two of the books created to target common variable misconceptions. Results revealed that students believed the books were engaging, and their self-perceived knowledge of variables increased after using the books. However, misconception rates remained comparable to prior research, highlighting the challenges in addressing misconceptions. Additionally, when students were asked to rank their prior programming language experience and

programming environments they have used, it was found that the students' computer science knowledge is behind the regional educational standards, emphasizing the need for educational material like CodeKids to teach computer science to young students. The findings suggest that while the storybook format used by CodeKids is a promising medium to teach children computer science, more research needs to be done to refine the design and develop tools that effectively address misconceptions.

Digital Storybooks to Teach Children Computer Science and Address Common Misconceptions

Thomas S. Deverin

(GENERAL AUDIENCE ABSTRACT)

Computers play a large role in our daily lives, and it's becoming increasingly important that younger students receive at least a foundational computer science education. CodeKids is an online platform designed to teach elementary and middle school students the basics of computer science through interactive digital books. These books use engaging stories and relatable examples to explain key ideas like how computers store information, make decisions, and repeat tasks. This study aims to see how children perceived the books and whether the books were effective at teaching concepts to children they typically misunderstand, such as variables (which store information in a computer program). We partnered with a local middle school STEM teacher who used two books in her 6th and 7th grade classes designed to teach children variables and address common misunderstandings regarding variables. After using the books, students reported enjoying the books and believed their understanding of variables improved. However, the results showed that many of the misunderstandings the books aimed to address were still present. It was also discovered that the children's knowledge of computer science was below regional educational expectations. These findings suggest that storybooks are a promising

format to teach children computer science, but more work is needed to ensure they effectively prevent misunderstandings. This research can help improve how we teach young students about computer science and make sure they develop a strong foundation in the topic.

Acknowledgments

I would like to express my gratitude to all of the undergraduate students whose work and dedication have made CodeKids a reality by developing the codebase and many of the books. I am also thankful for all of the teachers and students who have used the books and provided invaluable feedback to refine and improve the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

While research into computer science education in primary and middle school is expanding, there are still many questions surrounding the proper methodologies to use when teaching children these advanced topics at a young age. With computer science growing as a discipline, it's becoming more important for it to have the same ubiquity in classrooms as subjects like math or science. Many U.S. states, including Virginia, have adopted a standardized computer science learning curriculum for K-12 students, but there is difficulty in meeting these standards for many schools for various reasons, including a lack of teacher training, funding for resources, and limited curriculum integration [10]. It is important to overcome these challenges as research shows that teaching computer science to children as early as elementary school offers many benefits, including developing problem-solving and critical-thinking skills [8]. Additionally, incorporating computer science education at the K-8 level can expose more children from various demographics to computer science since courses in K-8 education are typically taught to entire grade levels instead of being offered as

electives.

However, with the limited research into elementary computer science education, it is still mostly unknown what approaches are effective at teaching computer science to young students. One of the most popular strategies used is visual block-based programming environments, such as Scratch. These environments are appealing and generally accepted for introducing advanced computer science concepts to young students. However, several studies show limitations regarding the cognitive understanding of children related to topics such as variables and loops when using block-based environments, and they can ignore the formation or even produce misconceptions [1, 6, 9, 15]. Our focus is to develop a medium of computer science education that is effective at teaching children advanced computational concepts and dispels common computer science misconceptions that elementary and middle school students often form.

1.2 CodeKids

CodeKids is a website developed to teach elementary and middle school students computer science through storybooks. The website is being continuously developed and hosts storybooks for computer science subjects ranging from bits and bytes to artificial intelligence. Currently, all books aimed at preventing and targeting misconceptions are marked as "Advanced". Meaning they are best for students in grades five through middle school. Figure 1.1 displays some of the advanced books currently on CodeKids with the following specifically created to target misconceptions

of computer science concepts: *Variables*, *Variable Data Types*, *Conditional Operators*, *Logical Operators*, *If Statements*, *Life of Moose*, *Flowcharts*, *For Loops*, and *While Loops*. More information about these books and the misconceptions they target will be discussed in Chapter 4.

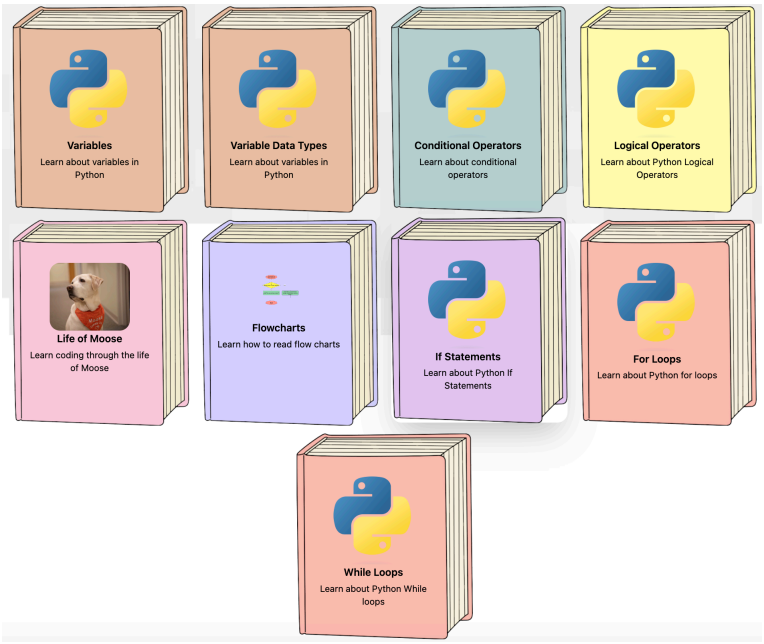


Figure 1.1: Collection of advanced computer science books available on the CodeKids website.

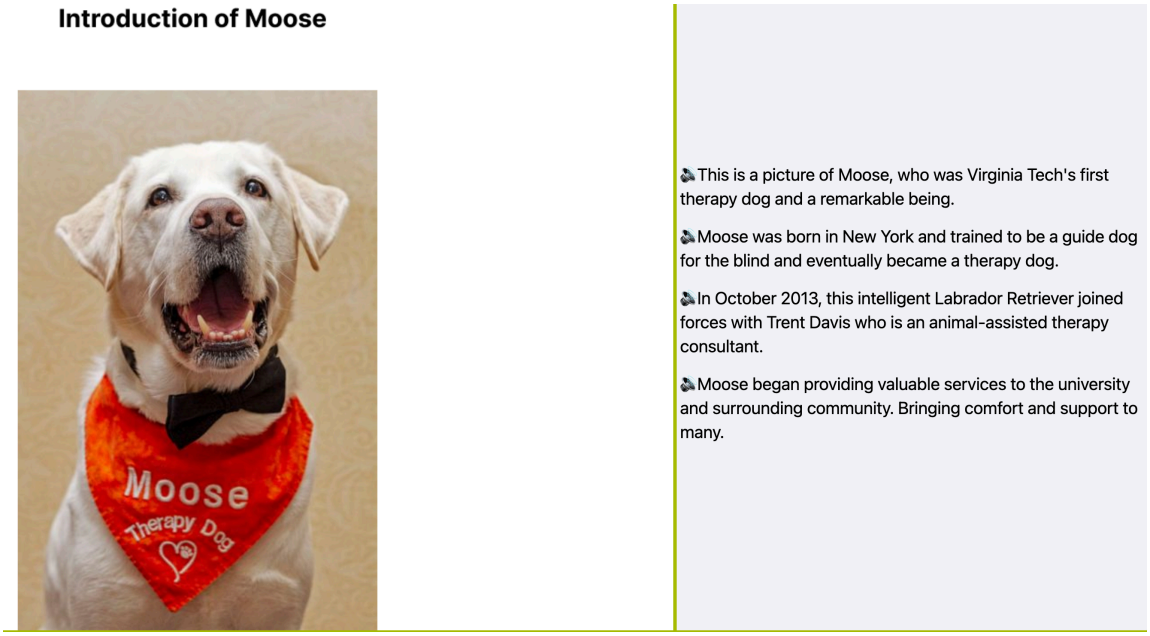
Instead of focusing purely on programming, these books incorporate stories and themes that children can relate to, fostering motivation and enjoyment while learning computational concepts. They also use interactive elements such as Python Tutor to visualize program execution step-by-step and questions to test their understanding of the material. Figure 1.2 demonstrates these elements in the *Life of Moose* book. Specifically, Sub-figure 1.2a shows the story elements depicting Moose's life, and Sub-figure 1.2b shows the interactive elements Python Tutor and question sets, which are

based on Moose's story and test knowledge on computational concepts. All of the books in CodeKids follow this structured design of having a relatable theme for children and interactive elements created related to the theme.

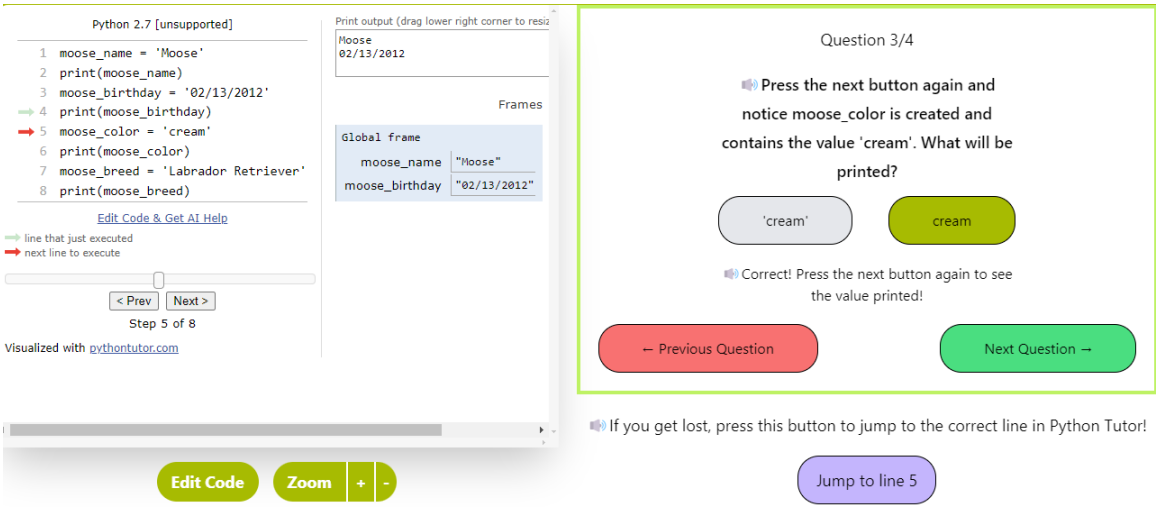
1.3 Research Questions

This research focuses primarily on the CodeKids books that were created to target and prevent common misconceptions children often form when learning computer science. For this reason, the following research questions are used to guide this study:

- **Does the storybook format provide children with a good format to learn computer science and improve their perceived self-knowledge towards computational concepts?**
- **How do misconception rates among students who used the digital books in CodeKids created to target misconceptions compare to rates reported in prior research, and what factors could contribute to any differences?**



(a) A story element in the *Life of Moose* book designed to engage and maintain the interest of children.



(b) Interactive elements in the *Life of Moose* book, including Python Tutor and a question set.

Figure 1.2: Interactive element from the *Life of Moose* book designed to teach children about variables.

Chapter 2

Review of Literature

2.1 Current Forms of CS Education in Elementary School

Block-based programming environments such as Scratch are widely used in elementary and middle school education to teach children programming. While these environments do have educational value, they tend to under-address or even at times exacerbate issues of computer science education in an elementary setting. For example, in a study comparing 10-11 year old students with one and a half years of block-based programming experience with students without programming experience, it was found that the two groups of students committed a similar number of syntax issues when writing text-based code [4]. This suggests that block-based programming merely delays syntactic problems. Additionally, studies analyzing the misconceptions regarding variables, conditionals, and loops formed by students often use block-based programming environments, indicating the environments, at best, don't prevent misconceptions from forming and, at worst, cause their formation [6, 15]. This is significant in the context of K-8 students since block-based envi-

ronments are commonly used to teach these students core computational concepts, which may be contributing to the development of misconceptions. Variables, in particular, are a difficult concept for children to comprehend and make up the most misconceptions listed in Section 2.3, both in the number of misconceptions and the proportion of students who form them. Luo et al. [9] found fourth-grade students using Scratch generally recognize variables as storage spaces for information, but struggle to operate on them and initialize their own variables.

Text-based languages, while not as popular for K-8 computer science education as block-based environments, are still used. Python, in particular, is known for being a great introductory text-based language. Despite this, Saito et al. [12] in their study with students aged 9-12 years old observed that students found Python to be intimidating to learn, and when asked about their willingness to learn programming, observed that student motivation to learn programming dropped slightly before and after using a text-based language. Additionally, many misconceptions have been observed in K-8 students learning from a text-based language as well. Žanko et al. [19], whose study uses Python to teach 10-11-year-old students variables, observed many misconceptions regarding the manipulation of variables, which will be discussed more in Section 2.3. This indicates that preventing the formation of misconceptions is not as simple as choosing text-based languages over block-based ones or vice versa. Rather, a new medium for computer science education needs to be created to target misconceptions.

Another form of computer science education is using Tangible User Interfaces (TUIs) to teach programming concepts. TUIs in computer science education allow students

to create programs with tangible objects and have been observed to be enjoyable for K-8 students to learn programming concepts from. Wang et al. [17] observed that students 6-8 years old are able to find errors more easily and correctly describe the actions of a program when presented with a TUI implementation. However, the effects of using a TUI become less significant as children get older, suggesting TUIs may be better suited for younger children [13]. Also, Merkouris and Chorianopoulos [11] observed that when 12-13 year old students learned programming concepts through the use of robotics first over computer-based programming, students only showed a minor performance improvement on a post-exam targeting the computational concepts sequence, repeat, and if-then-else compared to students who used Scratch first, suggesting that using robots as an introductory platform only has a neutral effect on learning. Lastly, there isn't any substantial research into how TUIs can either cause or prevent misconceptions regarding computational concepts, such as variables from forming.

2.2 Storybook Format

Storybooks provide a different approach to teach young students subjects and have been used as a method to teach children mathematics. In a literature review by Furner [3], it was found that using children's literature to teach math helps the subject feel less intimidating than traditional approaches and helps reduce anxiety among students. Some of the additional benefits noted in the literature review are advancing mathematical thinking, allowing for historical, cultural, and practi-

cal applications and connections, and lending itself to problem-solving and active involvement from the context of a story. Because of the cognitive relation between mathematics and computer science, it is likely that many of the advantages of using children's literature to teach mathematics are also applicable to computer science.

2.3 Misconceptions

This section lists the misconceptions found in previous studies. The misconceptions are categorized into three broad computational concepts: variables, conditionals, and loops. All misconceptions listed are from three studies that used either the programming languages Scratch or Python: Swidan et al. [15] who used Scratch with 7-17 year old students (mean age is 11 years old), Žanko et al. [19] who used Python with 10-11 year old students, and Grover and Basu [6] who used Scratch with 6th-8th grade students (specific age not specified).

2.3.1 Variable Misconceptions

From the studies reviewed, variables make up the most frequent misconceptions children form. The following is a list of variable misconceptions from Žanko et al. [19] and Swidan et al. [15]:

1. A variable is not stored in computer memory.

   Swidan et al. [15] found that 33.3% students believed variables were not stored

in computer memory. Instead, they believed variables were either not stored anywhere or only visible on the screen.

2. Assigning the expression instead of the calculated value.

```
a = a + 1
print(a)
```

Students believe the literal expression "a + 1" is assigned to the variable *a*. Žanko et al. [19] claim this misconception to be the most common for text-based programming and observed 21.94% of students held this misconception. Swidan et al. [15] also recorded this misconception in their study using Scratch, where 25.4% students held the misconception.

3. A variable can hold multiple values.

```
num = 5
num = 10
print(num)
```

Here, the student believes the variable *num* can hold many values and will say "5 10" is printed instead of "10". This was the second most common misconception Swidan et al. [15] observed in their study using Scratch, with 42.9% of students holding this misconception.

4. When reassigning the variable's value, it assigns the sum of the previous and new values.

```
x = 5
```

```
2                    x = 10
3                    print(x)
```

In this case, the student believes reassigning a variable has the effect of adding to its current value. In the example above, the student believes "15" is printed instead of "10". Žanko et al. [19] found 18.8% of students held this misconception.

5. Variables assignment works in opposite directions.

```
1                    num1 = 5
2                    num2 = 10
3                    num1 = num2
4                    print(num1, num2)
```

Here the student believes *num1* and *num2* will swap values, so the program will print "10 5" instead of "10 10". This misconception was found by Swidan et al. [15] who observed that 3.9% students held this misconception.

6. The natural-language semantics of a variable's name affects the values assigned to it.

```
1                    big_number = 50
2                    small_number = 10
3                    big_number = small_number
4                    print(big_number, small_number)
```

In this case, the student incorrectly believes that *big_number* cannot be assigned the value of 10 due to the semantic meaning of the variable's name.

They believe either “50 10” or “50 0” will be printed. Swidan et al. [15] observed 20.6% of students held this misconception.

7. Using the symbolic name of the variable instead of its value.

```
first = 100
second = 1
print(first, second)
```

Here, the student believes that when a variable is present in a print statement, the variable’s name will be printed instead of its value. In the example above, the student believes “first second” is printed instead of the variables’ values. Žanko et al. [19] found 7.14% of students held this misconception.

8. Data type mistake where students will print the value of a variable instead of a string that happens to be a variable’s name.

```
number = 100
print('number')
```

Similar to the previous misconception, except this time the student doesn’t recognize that the value in the print statement is the string data type. The student believes “100” is printed instead of “number”. Žanko et al. [19] found 33.68% held this misconception.

9. Using the first (or previously) assigned variable value.

```
num1 = 10
num2 = 5
num2 = num1
```

```
4                    print(num1, num2)
```

In this case, the student doesn't understand that code executes sequentially and doesn't recognize that the value of a variable is changing. Žanko et al. [19] believes this problem occurs mostly when a variable is reassigned to something that isn't a number. In the example above, *num2* isn't reassigned to a number but is instead reassigned to the variable *num1*. The student believes "10 5" will be printed instead of "10 10". Žanko et al. [19] observed that 18.88% of students held this misconception.

10. Difficulty understanding the sequential execution of code.

```
1                    num1 = 0
2                    num2 = 0
3                    total = num1 + num2
4                    num1 = 5
5                    num2 = 10
6                    print(total)
```

Similar to the previous misconception, the student doesn't understand that code executes sequentially and believes the program above will print "15" instead of "0". They ignore the fact that *num1* and *num2* are both 0 when *total* is assigned, and instead use the values of *num1* and *num2* after *total* is assigned. Swidan et al. [15] observed this to be the most frequent misconception in their study, with 56.2% of students holding it.

11. Swapping variables

```
num1 = 5
num2 = 10
num3 = num1
num1 = num2
num2 = num3
```

For this misconception, Žanko et al. [19] found that students ignore the creation of the variable *num3* and that variable swapping would happen simultaneously instead of sequentially. They found 13.01% of students held this misconception.

2.3.2 Conditional Misconceptions

Conditional misconceptions are far fewer in studies. It's possible that children are less likely to form misconceptions around this computational concept, or there isn't enough research investigating the misconceptions for this particular concept. Swidan et al. [15] noted this, stating that misconceptions related to control statements were the least common in their study.

1. False condition ends a program if there is no else.

```
x = 1
if x == 2:
    print('True')
print('False')
```

Here the student believes nothing is printed because there is no else branch.

Swidan et al. [15] observed 7.4% of students held this misconception.

2. OR operator evaluates to false when both operands are true. Here, the student believes the OR operator works like the XOR operator, so the expression *True or True* would evaluate to false instead of true. Grover and Basu [6] observed that 6.5% of students held this misconception.

2.3.3 Loop Misconceptions

Loops are another concept that children often form misconceptions about. The study from Grover and Basu [6] focuses primarily on identifying student misconceptions regarding loops and Swidan et al. [15] also observes several loop misconceptions.

1. Adjacent code executes when a loop is running.

```
1  number = 0
2  for x in range(5):
3      number = number + 1
4  print(counter)
```

   Here, the student believes the print statement, which is adjacent to the loop, is part of the loop. A student with this misconception will claim "1 2 3 4 5" will be printed instead of "5". Swidan et al. [15] found 33.3% of students held this misconception.

2. A loop terminates as soon as the condition becomes false instead of continuing until the condition is checked again.

```
x = 1
while x < 3:
    x = x + 1
    print(x)
```

In this case, the student believes that as soon as x becomes three, the loop will terminate without finishing its current execution. Therefore, the student believes only "2" is printed instead of "2 3". Swidan et al. [15] observed that 10.5% of students held this misconception.

3. Instead of executing the loop in sequence several times, the student believes each line of code is executed on its own.

```
for x in range(3):
    print("Start")
    print("Stop")
print("Done")
```

Grover and Basu [6] described this misconception as the "grouping error", where students group each line of code and execute each line the number of times the loop will execute instead of executing the code sequentially. In the example above, the student would claim the code prints "Start Start Start Stop Stop Stop Done". Grover and Basu [6] observed that 8% of students held this misconception.

4. The number of times a loop repeats using preassigned variables.

```
numberOfTimes = 3
```

```
2                   count = 0
3                   for x in range(numberOfTimes):
4                       count = count + 2
5                       print(count)
6                   print("Last " + str(count))
```

For this misconception, students either add or miss an iteration of the loop when using a preassigned variable. In the example above, the student would say the program prints either "2 4 6 8 Last 8" or "2 4 Last 4". The correct answer is "2 4 6 Last 6". Grover and Basu [6] observed 7.8% of students added an iteration and 3.9% missed an iteration. They also noted that only two-thirds of students understood that the value of *count* changes while *numberOfTimes* does not. It's also important to note that it's not an issue where a student may not understand how Python's range function works (like it is used in the example above) because Grover and Basu [6] used Scratch in their study.

Chapter 3

Methodology

3.1 Creation of CodeKids

CodeKids was created to be an educational website to teach children computer science. The primary target audience are elementary and middle school students with books categorized by "Beginner", "Intermediate", and "Advanced" levels. It contains books that cover a wide breadth of computer science topics from foundational computational concepts such as variables to more advanced ideas related to artificial intelligence. In general, "Beginner" books are intended for kindergarten to 2nd grade, "Intermediate" books are intended for 3rd to 4th grade, and "Advanced" books are intended for 5th to 8th grade, with some expected overlap between levels.

The website was built using the React JavaScript library for the front-end and Python libraries, including FastAPI and Uvicorn for the back-end. Additionally, the Prisma ORM was used to create a schema defining models for the PostgreSQL database. Users can create student or teacher accounts, and interaction data is recorded in the database to be used for future research.

Creating a book in CodeKids is a multi-step process that Figure 3.1 outlines. The

first step is to review the K-12 computer science learning standards to identify the key learning outcomes for students in specific grades. The document outlines in great detail the computer science intended learning outcomes for students starting in kindergarten [16]. Next, we determine themes that are relatable to children and could effectively convey computational concepts to children while also meeting the learning standards. The books are then designed in Figma [7] and then implemented into the CodeKids website. Once implemented, the books are reviewed by K-8 teachers, and suggestions they make are used to inform design improvements. Lastly, students use the books and provide survey feedback, which we use to further improve the books. This iterative process ensures that CodeKids books are usable in classrooms and enjoyable for young students.

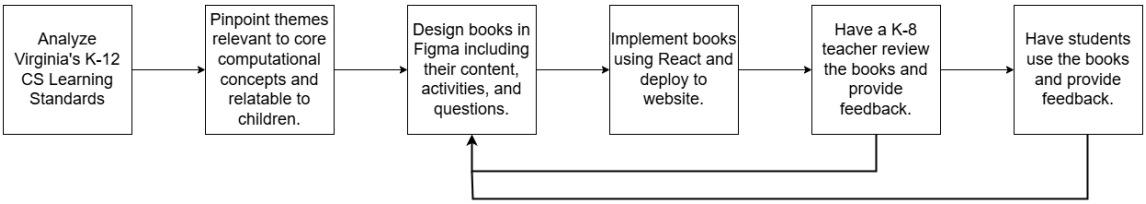


Figure 3.1: Chart illustrating the key steps in making a book for CodeKids.

3.1.1 Alignment with the State Computer Science Curriculum

Because these books are meant to be used in a formal classroom environment, the content of the books must align with the computer science curriculum of K-8 schools. We referenced the *Computer Science Standards of Learning for Virginia Public*

Schools, which outlines the standards of computer science learning for students in kindergarten through eighth grade in Virginia [16].

All of the books created for this study were specifically designed to align with learning standards for children in the fifth to seventh grades. In sixth grade, particularly, the standards dictate that students should begin to use text-based programming languages and are expected to "implement algorithms that include conditional control structures and collection of numeric data" and "read and write programs that initialize Boolean, integer, and decimal number variables" [16]. Additionally, in the seventh grade, students are expected to "read and write programs that contain nested conditionals and nested loops" [16].

With that said, not all of our books align with the standards. For example, some artificial intelligence books were designed for lower grade levels, despite artificial intelligence not showing up until 6th grade in the learning standards. We made this decision so that the content of our website isn't completely constrained by the learning standards, and children using the website outside class could learn a concept they might be interested in.

3.1.2 Design of the Books

All books are designed with a story or theme in mind to help make the content more relatable for children and make advanced computational concepts less daunting. For example, the *Life of Moose* book tells the story of Virginia Tech's late therapy dog Moose, and themes from other books include farm animals, football scores, and

restaurant food menus. All of which have questions and Python Tutor examples that use code related to these themes.

After conceptualization of the books, we used Figma to create the design of different pages of the books, including their computer science content and interactive elements. Figma was chosen due to its robustness as a high-fidelity wireframing tool. This approach allowed us to rapidly make design decisions and iterations of the designs before programming the content onto the main website. Figure 3.2 showcases some of these designs for the *Variables* book. Descriptions of the concepts students are learning are on the right page of the book, while images and interactive elements are on the left side. Creating prototypes like this helped visualize the user experience before taking the time to implement the books on the website.

3.1.3 Teacher and Student Review

Once the books were implemented, a K-8 teacher reviewed each of the books and provided feedback on its content. It is important to get the opinion of a teacher on the books, since a teacher would have a deeper understanding than researchers of how students will interact with and perceive educational material. The reviewer gave suggestions on various aspects of the books, including the visuals, vocabulary, and exercises. They also noted where each book aligned with the computer science learning standards. Any recommended changes were promptly implemented.

After the books were used in the classroom, students completed surveys to share their opinions on the books. The survey included questions asking students what

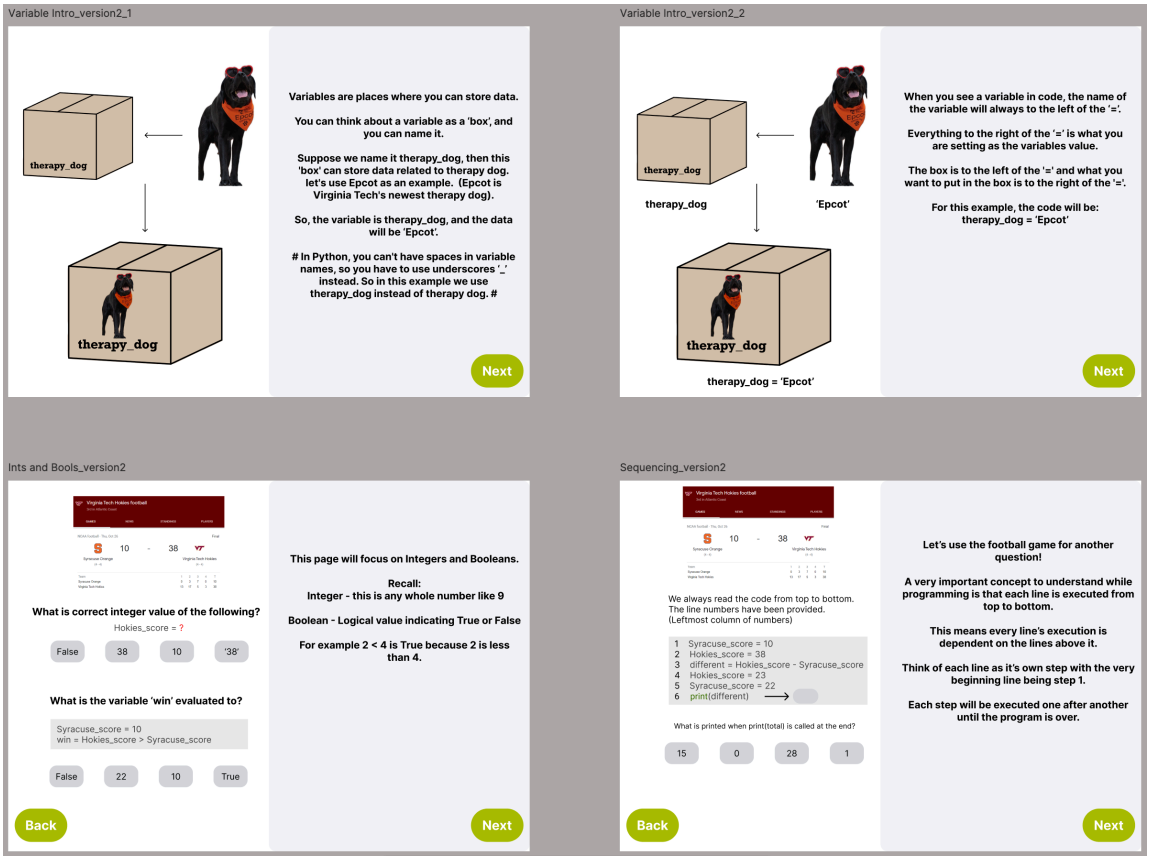


Figure 3.2: Figma prototypes of pages in the CodeKids book *Variables*.

they thought of the books' activities and how difficult they thought the books were. Students responded to these types of questions on a 1-5 scale and had the option to explain their responses. This data is also used to inform book redesign.

3.1.4 Book Editor

A feature is currently being implemented to allow teachers to create their own books. This feature is still early in its development, but the tool is designed to offer pre-

defined components that enable teachers to create custom books tailored to their students' learning needs. This feature has also helped us streamline the process of implementing and publishing books once they have been designed in Figma.

Figure 3.3 shows the book editor in its current form and its key functionalities. At the top is a drop-down menu where a component can be selected. In this case, the component "Image Question" is selected, which allows someone to create a question with an image to accompany it. There are various other components to choose from, including "tutor", "Image", and "Fill In The Blank". Once a component is selected, the user can configure the aspects of the component in the editor. The editor uses a form input by default for users to enter details for the component, and users can switch to a JSON view as well. To the left in the book editor is an area to manage pages in the book. Pages can be added, deleted, and moved using the accompanied buttons. Users can also see a preview of the book and change its details, such as the title and blurb. To the right is an area where the text of the page is written. Lastly, the bottom gives a preview of the component and how it will look in the book.

While functional, the book editor has limitations that need to be fixed before advertising the feature to teachers. For one, the form input could still be daunting to use for some inexperienced users. To accommodate this, help icons should be implemented for each component to help new users learn how to use each component. Secondly, a lot of components are "single-use" components, meaning they only work for a specific page in a specific book. This is a relic of the previous way books were created, where each page was programmed in React. Work will be done to convert these pages to use more generalized components like the "Image Question" compo-

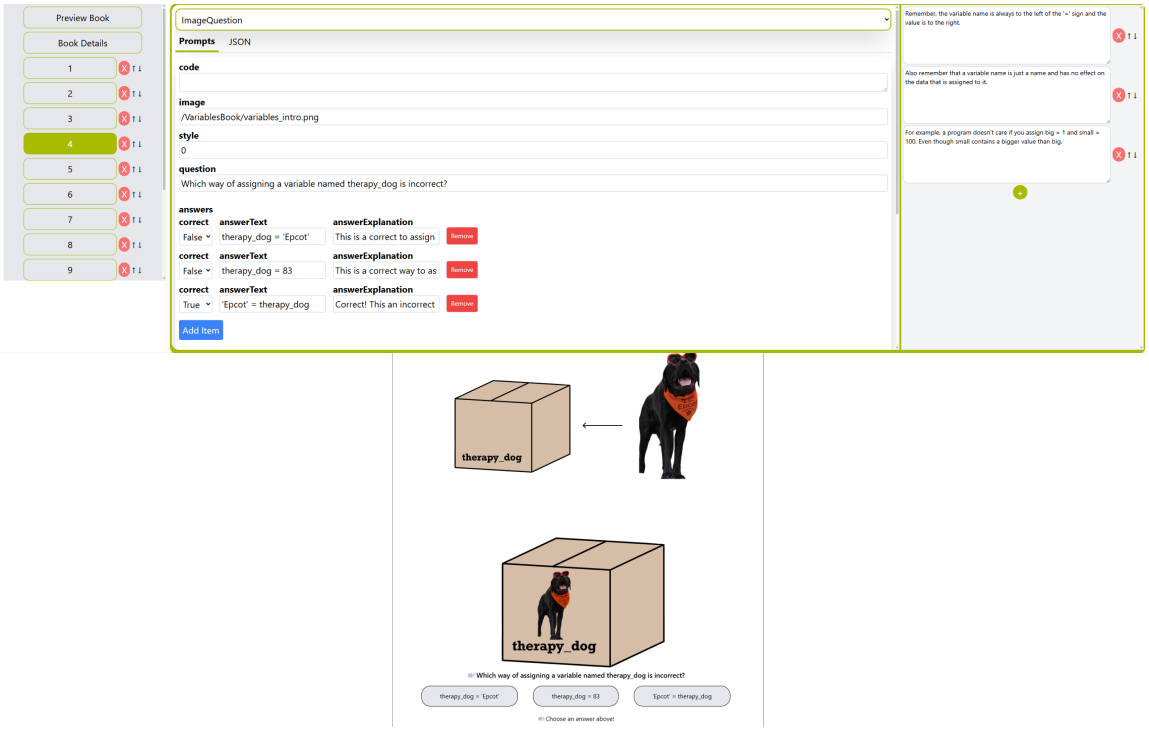


Figure 3.3: Book editor interface for creating custom books in CodeKids.

nent. Lastly, there is no save button, and any changes are immediately sent to the database. This is not user-friendly since a user may want to discard changes they have made.

3.2 Classroom Instruction

To best represent a real classroom environment, the books were given to a STEM teacher at a local middle school. The teacher used the books during their regular classroom instruction and was aware of the misconceptions the books were created to address. The students taught were in the sixth and seventh grades, with the

children's ages ranging from 11 to 14 years old. The students received instruction during normal class time for two weeks. The IRB approval letter can be referenced in Appendix C.

3.3 Examination

After using the books, the teacher administered a paper-based multiple-choice and matching examination of the material in the books. This exam was administered after the two weeks of instruction at the end of November 2024. The examination contained questions targeting the misconceptions in the books and additional general knowledge questions. The questions were designed by the author and follow similar themes to the questions present in the Codekids' books. Before administration, the teacher was encouraged to provide feedback or suggestions about the exam, but no feedback was given. A total of 115 students took the exam in the 6th and 7th grades. The exam can be referenced in Appendix A.

An important aspect to note is that the exam wasn't graded in a traditional way. The teacher notified us that they are unable to assign grades on the traditional A-F scale and can only give their students a passing or failing mark that is dependent only on their completion of the exam. This limitation is discussed further in Chapter 6 as it could have affected the students' motivation to perform well on the exam.

After the exams were received from the teacher, the results were recorded in an Excel spreadsheet for statistical analysis. Since this study compares the proportion of

students who answered the misconception questions in our study with other studies, a two-proportion z-test was conducted on each question targeting a misconception. This test was appropriate since the sample size is large enough to assume normality and each observation is independent. A statistically significant result would indicate that the percentage of students who chose the misconception answer option was either significantly higher or lower compared to previous studies. In cases where students left a question blank or left multiple answers, their responses were excluded from the analysis for that particular question.

Lastly, an independent samples t-test was conducted to determine if there was a significant difference in exam scores between the 6th and 7th grade students. The test compared the mean scores of both groups to observe if any difference is statistically significant. Python was used to conduct this test and create visualizations. Assumptions of normality and homogeneity of variance were checked using the Shapiro-Wilk test and Levene's test, respectively.

3.4 Survey

After using the *Variables* and *Variable Data Types* books, the students were given a survey to take. This survey was completed online using Google Forms. The survey gathered general information about the students, such as their previous programming experience, interest in programming, and their opinion of the books. Specifically, they were asked whether they enjoyed the activities in the books and if they believed that the tools utilized in the books helped them learn the material. Lastly, they

were asked to rate their perceived knowledge of the subject before and after using the books.

Results from the survey were exported to an Excel spreadsheet, where visualizations were created for each question. Additionally, statistical analysis in Python was conducted to examine potential correlations between the students' age groups and the questions asked. Based on the distribution of ages, students were grouped into three age groups: 11, 12, and 13-14 years old. Since the survey data consists of ordinal categorical data, the Spearman correlation test was used because it is a non-parametric test that measures the strength and direction of relationships between ranked variables. Meaning, it could be determined if older or younger students were more likely to exhibit a certain characteristic, such as having more programming experience.

Lastly, statistical analysis was done in Python to determine if students' perceived knowledge of variables increased significantly. This was done by calculating the difference of each student's post and prior perceived knowledge and performing a Wilcoxon Signed-Rank test. This test was chosen because the data is non-parametric and represents paired ordinal responses, making it more appropriate for analyzing changes in ranked perceptions.

Chapter 4

Digital Books

To teach elementary and middle school students computer science concepts, several digital storybooks were created on the CodeKids website. The books discussed in this chapter were created not only to teach the computational concepts of variables, conditionals, and loops, but also to target many of the misconceptions from Section 2.3. After giving an overview of the material, the students are given both questions to test their understanding and Python Tutor examples for them to click through.

Each book is typically 10-15 pages long and is meant to be completed by students in one or two class sessions. The book utilizes themes such as football scores, restaurant menus, and farm animals to help make learning computer science more relatable and less abstract to the students. As mentioned by Furner [3] in his literature review, using such themes can help young students form cultural applications and connections while also lending itself to problem-solving through the context of a story.

In this chapter, any reference to students and the proportion of those who held a misconception comes from any of the following samples from previous literature:

1. Swidan et al. [15] - students aged 7-17 years (mean age 11) who used Scratch.

2. Žanko et al. [19] - students aged 10-11 years who used Python.

3. Grover and Basu [6] - students in the 6th-8th grade who used Scratch.

4.1 Variables Books

Variables constitute many of the misconceptions elementary students form, as variables are a complex concept for children to learn. Children generally understand them as storage spaces but struggle to operate on them and modify code to manipulate them [9]. The following books were created to directly target the variable misconceptions listed in Subsection 2.3.1.

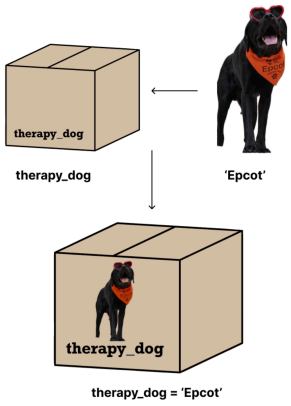
4.1.1 Variables

The *Variables* book introduces students to the general concept of variables in programming. The content of the book covers essential knowledge regarding variables, including what they are, how they store values, and how to use them in code. However, instead of merely describing facts about variables in computer science, the book uses themes and imagery that the children can relate to as well as interactive elements. Examples of these can be seen in the sub-figures of Figure 4.1. The rest of the books described in the chapter use a similar style, though themes may vary between books and topics.

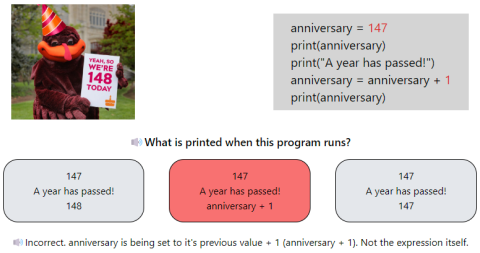
Figure 4.1 showcases several examples of the themes, imagery, and interactive ele-

ments in the *Variables* book. Sub-figure 4.1a uses the therapy dog Epcot to teach students that variables act as storage spaces. Sub-figure 4.1b shows Virginia Tech's anniversary being used to demonstrate how variables are assigned values. Sub-figure 4.1c uses football scores to demonstrate variable assignment and the sequential execution of code. Lastly, sub-figure 4.1d shows the usage of Python Tutor to demonstrate variable assignment in Python.

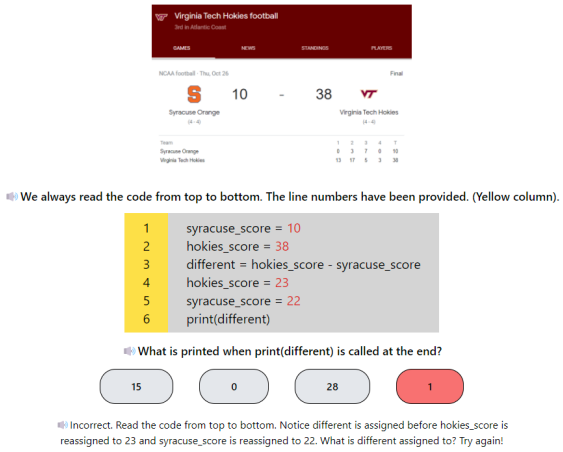
Table 4.1 lists out all the misconceptions present in this book. Each misconception is paired with specific examples from the book designed to address the student's misunderstanding.



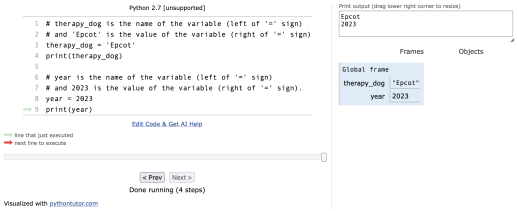
(a) Virginia Tech's therapy dog Epcot is used to show variables as storage spaces.



(b) Virginia Tech's anniversary is used to demonstrate variable assignment.



(c) Virginia Tech's football score is used to show that code is executed sequentially.



(d) Python Tutor used to teach variable assignment.

Figure 4.1: The *Variables* book uses themes relatable to children and interactive elements such as Python Tutor and questions to learn about variables.

Table 4.1: Misconceptions targeted in the *Variables* book.

Misconception	Misconception Type	Misconception Example	Discussion
A variable is not stored in the computer memory or is just a visual on the screen.	**Variable** - the student doesn't know a variable is stored in the computer's memory.	number = 10 Where is the variable *number* stored? Student Answer: Swidan et al. [15] noted common answers were "Variable is not stored anywhere" and "Variable is only visible on the screen". Correct Answer: "In the computer's memory."	The student believes the variable is either not stored in the system's memory at all or is just part of the display. The given book example explicitly tells students that variables are stored in the computer's memory. Swidan et al. [15] observed 33.3% of students held this misconception.

Book Example

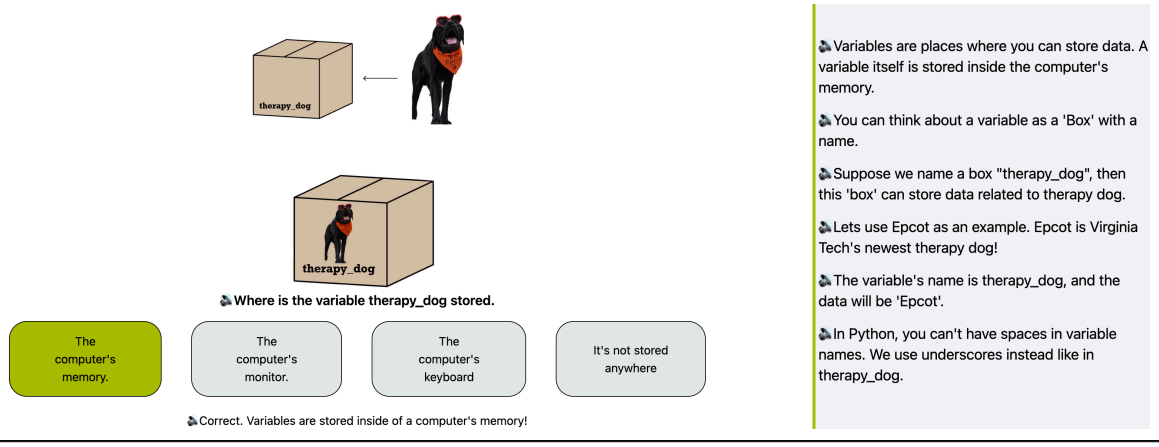


Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
Assigning an expression instead of a value to a variable.	**Variable Assignment** - the student misunderstands what is assigned to a variable when the assignment is an expression.	a = 10 a = a + 1 print(x) Student Answer: "a + 1" Correct Answer: "11"	The student thinks that the expression is assigned to the variable instead of the solution to the expression. The book example uses Virginia Tech's anniversary and adds 1 to its value. Swidan et al. [15] found 25.4% and Žanko et al. [19] found 21.94% of students hold this misconception.

Book Example


```
anniversary = 147
print(anniversary)
print("A year has passed!")
anniversary = anniversary + 1
print(anniversary)
```

What is printed when this program runs?

147
A year has passed!
148

147
A year has passed!
anniversary + 1

147
A year has passed!
147

Incorrect. anniversary is being set to it's previous value + 1 (anniversary + 1). Not the expression itself.

We can also use a variable's old value to assign itself a new value.

In this example, we are reassigning anniversary to be it's old value plus 1.

Read through the code and answer the question!

Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
A variable can hold multiple values.	**Variable** - the student doesn't understand that a variable can hold only one value at a time.	num = 5 num = 1 print(num) Student Answer: "5 1" Correct Answer: "1"	The student believes a variable can hold many values instead of just one. The book example reassigns the variable *anniversary* and asks students the variable's value. Swidan et al. [15] found 42.9% of students hold this misconception.

Book Example

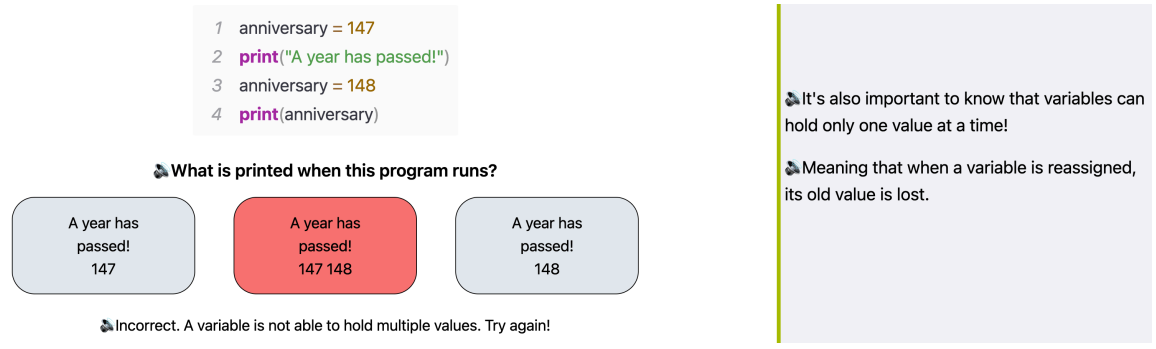


Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
Summation of a variable's value when it's reassigned.	**Variable Assignment** - the student doesn't understand how variable reassignment works.	x = 5 x = 10 print(x) Student Answer: "15" Correct Answer: "10"	The student thinks reassigning a variable will sum the variable's old value with the new value. The book's example reassigns *anniversary* and asks what the new value is. Python Tutor is also used to show variable reassignment. 25% of students hold this misconception [19].

Book Examples


```
anniversary = 147
print(anniversary)
print("A year has passed!")
anniversary = 148
print(anniversary)
```

🔊What is printed when this program runs?

147
A year has passed!
148

147
A year has passed!
147

147
A year has passed!
295

🔊Incorrect. Assigning a variable a new value DOES NOT add it with the old value. The variable is simply assigned the new value. Try again!

🔊Variables can be assigned different values throughout a program.

🔊When a variable that already has a value is assigned a new value, the old value is lost.

🔊Think of your age as a variable. If you are 8 years old now, on your birthday your old age will be lost and your new age will be 9!

🔊Try out the examples and exercises.

Table 4.1: Misconceptions targeted in the *Variables* book

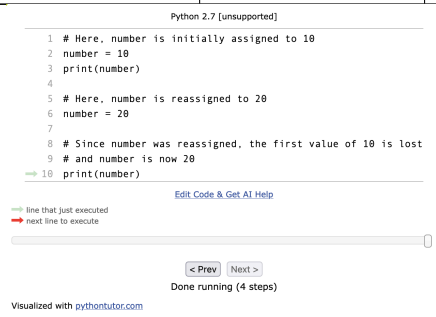
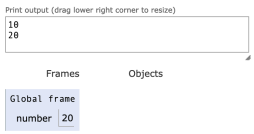
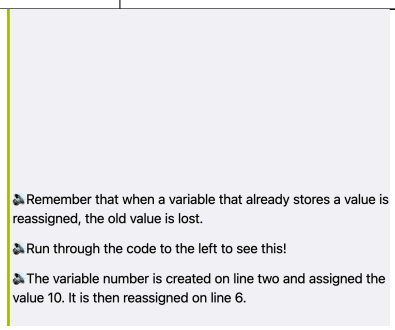
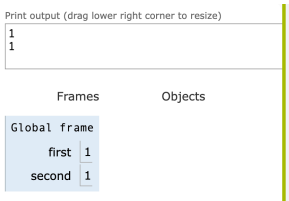
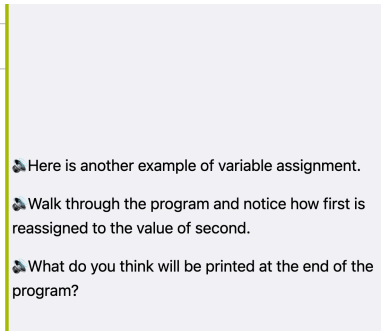
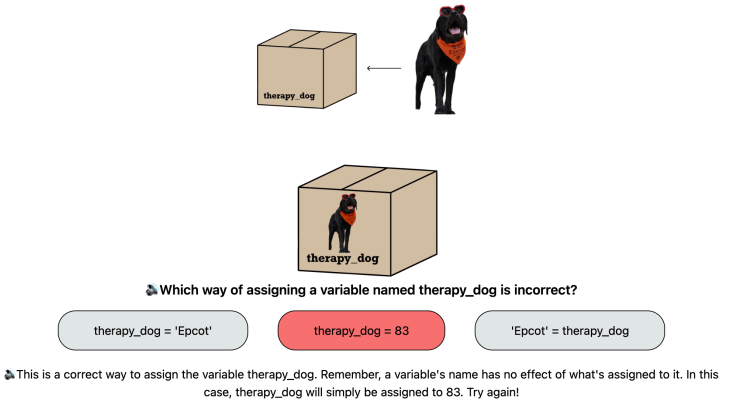
Misconception	Misconception Type	Misconception Example	Discussion
			
Variable assignment works in opposite directions.	**Variable Assignment** - the student believes the variable to right of the "=" sign can also be assigned.	num1 = 5 num2 = 10 num1 = num2 print(num1, num2) Student Answer: "10 5" Correct Answer: "10 10"	The student believes that variable assignment will work in the opposite direction and swap the values of the two variables. The book example uses Python Tutor to show an example. 3.9% of students hold this misconception [19].
Book Examples			
			

Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
Natural-language semantics of a variable affects the variable's value.	**Variable Assignment** - the student believes a variable's name affects what can be assigned to it.	big = 50 small = 10 big = small print(big, small) Student Answer: "50 10" or "50 0" Correct Answer: "10 10"	The student believes the variable small should be the smaller value 10, incorrectly inferring the value based on the variable's name. The book example shows the variable *therapy_dog* can be assigned a number. Up to 20.6% of students hold this misconception [19].

Book Example



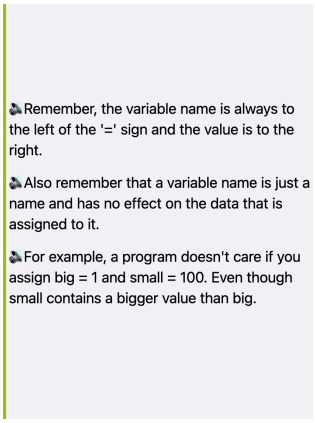


Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
Using the first or previously assigned value of a variable.	**Code Sequence** - the student doesn't understand that code is sequential and ignores a variable reassignment.	num1 = 10 num2 = 5 num2 = num1 print(num1, num2) Student Answer: "10 5" Correct Answer: "10 10"	The student thinks a variable will retain the value it's first assigned to. They don't understand that code runs sequentially. Žanko et al. [19] also believes this stems from a variable being assigned something other than a number (e.g. another variable). The book example uses Python Tutor to show the variable *first* has changed when printed. Up to 28.5% of students hold this misconception [19].
Book Example			

Table 4.1: Misconceptions targeted in the *Variables* book

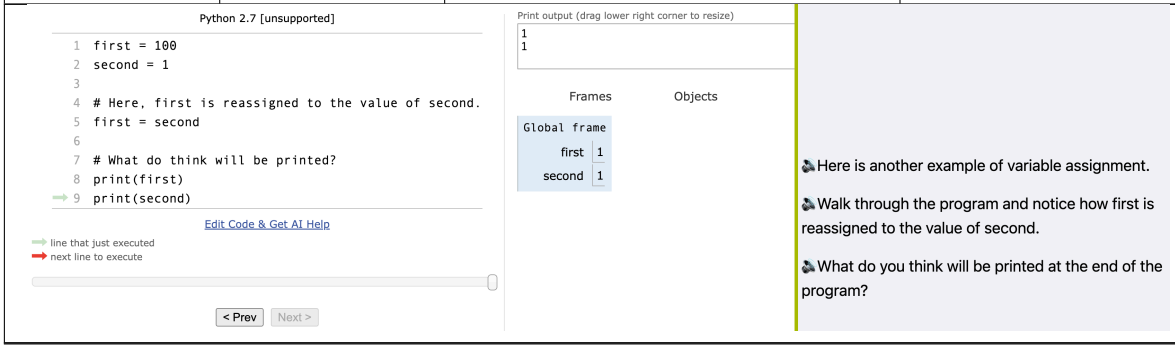
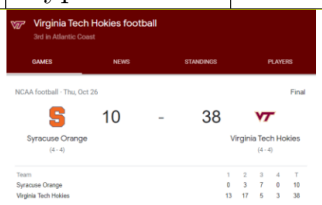
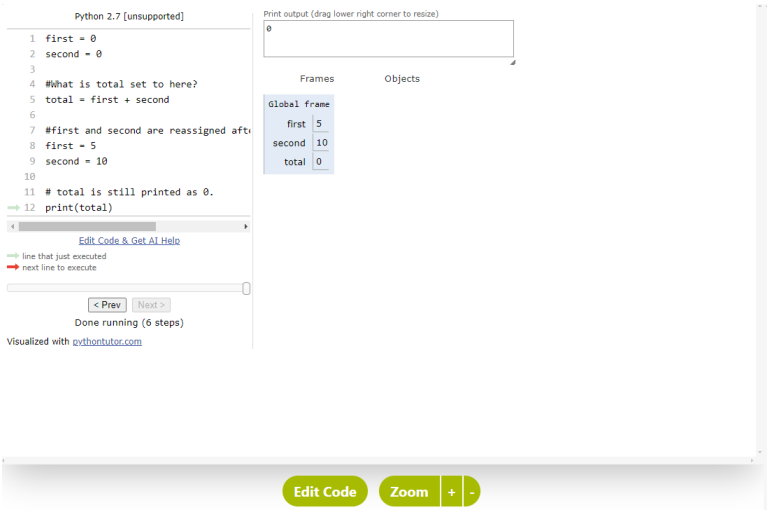
Misconception	Misconception Type	Misconception Example	Discussion
			

Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
Code doesn't execute sequentially.	**Code Sequence** - the student doesn't understand that code execution is sequential and assigns the variable to the sum of proceeding variables.	num1 = 0 num2 = 0 total = num1 + num2 num1 = 5 num2 = 10 print(total) Student Answer: "15" Correct Answer: "0"	The student doesn't recognize that code is executed sequentially and thinks *total* will be assigned to the sum of *num1* and *num2* after their reassigned. The book example uses football scores to calculate the difference in scores and Python Tutor to show a similar example. Swidan et al. [15] found this to be the most common misconception at 56.2%. Žanko et al. [19] also found sequencing mistakes to be a common misconception.
Book Examples			

Table 4.1: Misconceptions targeted in the *Variables* book

Misconception	Misconception Type	Misconception Example	Discussion
		 We always read the code from top to bottom. The line numbers have been provided. (Yellow column). 1 syracuse_score = 10 2 hokies_score = 38 3 different = hokies_score - syracuse_score 4 hokies_score = 23 5 syracuse_score = 22 6 print(different) What is printed when print(different) is called at the end? 15 0 28 1 Incorrect. Read the code from top to bottom. Notice different is assigned before hokies_score is reassigned to 23 and syracuse_score is reassigned to 22. What is different assigned to? Try again!	Let's use this football game's score to answer the next question! A very important concept to understand while programming is that each line is executed from top to bottom. This means every line's execution is dependent on the lines above it. Think of each line as its own step with the very beginning line being step 1. Each step will be executed one after another until the program is over.
			Click through the code to the left and think about how the code is executed from top to bottom. Even though first is reassigned to 5 and second is reassigned to 10, total was calculated beforehand.

4.1.2 Variable Data Types

The *Variable Data Types* book introduces children to the idea of variables storing different kinds of data. The book teaches children three fundamental data types: integers, booleans, and strings. Figure 4.2 shows an example of one of the book pages teaching children about the different data types. Specifically, it demonstrates integers using a jersey number, strings using a therapy dog's name, and booleans by showing that a variable *sky_is_blue* would be true. These familiar, real-world examples are meant to help children grasp the idea of these different data types.

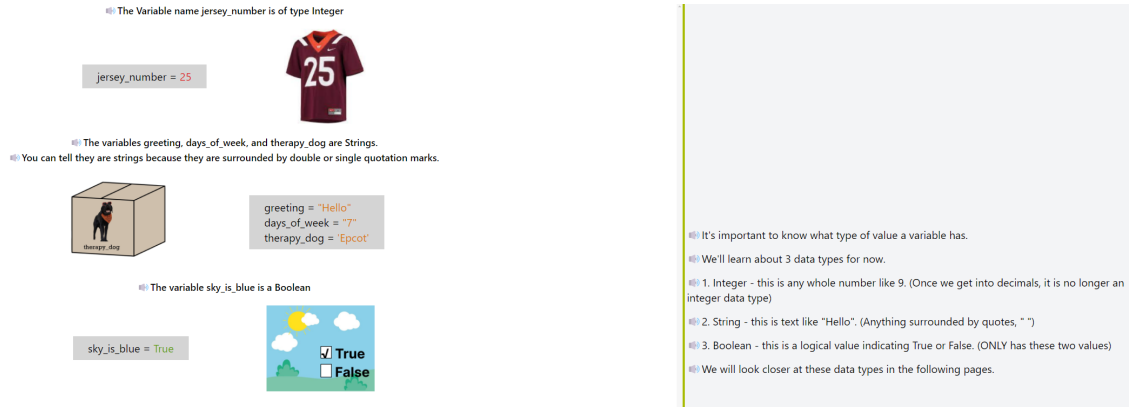


Figure 4.2: A page from the *Variable Data Types* book teaching children about integers, strings, and booleans using relatable themes to including sports jerseys, dogs, and weather conditions.

This book addresses two misconceptions, both of which are related to one another and can be referenced in Table 4.2. The first misconception is when a student fails to recognize the difference between a string value and a variable that happens to have the same name as that string. The second misconception is the opposite, and is when students mistake a variable name in a print statement to be a string.

Table 4.2: Misconceptions targeted in the *Variables Data Types* book.

Misconception	Misconception Type	Misconception Example	Discussion
Printing a string that happens to be a variable name prints the variable's value.	**Data Type** - the student doesn't notice the difference between a variable and a string with the same name.	number = 100 print("number") Student Answer: "100" Correct Answer: "number"	The student doesn't recognize that the value inside the print statement is a string, and not the variable with the same name. The book example explicitly targets this and color codes the string to help students notice it's different from the variable. The Python Tutor example encourages students to pay close attention to what is in the print statement. Žanko et al. [19] observed 33.68% of students held this misconception.
Book Examples			

Table 4.2: Misconceptions targeted in the *Variables Data Types* book.

Misconception	Misconception Type	Misconception Example	Discussion
		 anniversary = 147 print("A year has passed!") anniversary_2 = "148" print('anniversary_2') 🔊What is printed during this program? A year has passed! 148 A year has passed! anniversary_2 A year has passed! "148" 🔊Incorrect. Pay close attention to what's in the final print statement. Try again!	🔊Take a moment to think about his question. 🔊Pay close attention to what's in the print at the end. 🔊Remember that anything surrounded by single or double quotation marks are Strings.
		Python 2.7 [unsupported] 1 virginia_tech_age = 151 2 3 # Pay close attention to what's in the print statement. 4 # Is it actually the variable virginia_tech_age? 5 print("virginia_tech_age") Edit Code & Get AI Help line that just executed next line to execute < Prev Next > Done running (2 steps) Visualized with pythontutor.com Print output (drag lower right corner to resize) virginia_tech_age Frames Objects Global frame virginia_tech_age 151	🔊Look that the code to the left. 🔊Pay close attention to what is actually in the print statement. 🔊What will be printed at the end of the program? 🔊At first, you may think it will be 151 since that is the value of virginia_tech_age. 🔊However, notice that what is actually inside the print statement is the String "virginia_tech_age". 🔊So, the String value "virginia_tech_age" is actually in the print statement, not the variable virginia_tech_age.

Table 4.2: Misconceptions targeted in the *Variables Data Types* book.

Misconception	Misconception Type	Misconception Example	Discussion
Using the variable's symbolic name instead of its value.	**Variable** - the student doesn't understand that printing a variable prints its value. **Data Type** - the student thinks the variable name in a print statement is a string.	first = 1 second = 100 print(first, second) Student Answer: "first second" Correct Answer: "1 100"	The student thinks printing a variable will print the variable name. This may be because they view the variable name as a String. The book example has an incorrect answer that prints the name of the variable *therapy_dog_left*. Žanko et al. [19] found 7.14% of students held this misconception.

Book Example

```
therapy_dog_left = "Derek"
therapy_dog_middle = 'Epcot'
therapy_dog_right = "Josie"
print(therapy_dog_left)
```

What is printed at the end of this program?

"Derek" | therapy_dog_left | Josie | Derek

Incorrect. therapy_dog_left is a variable. The print statement will print the value of the variable.

When a String is printed, the quotation marks are not included.

So, print("Hello") will print Hello.

4.1.3 Life of Moose

Life of Moose is a book that tells the story of Moose, a Virginia Tech therapy dog. This book is slightly different from the others as it doesn't teach children new concepts. Instead, this book reinforces previously learned material by requiring students to answer questions that target the variable misconceptions from both the *Variables* and *Variable Data Types* books. While reading the book, children use Python Tutor to click through code and answer questions about Moose's life. Table 4.3 shows examples of questions in the book that target various variable misconceptions.

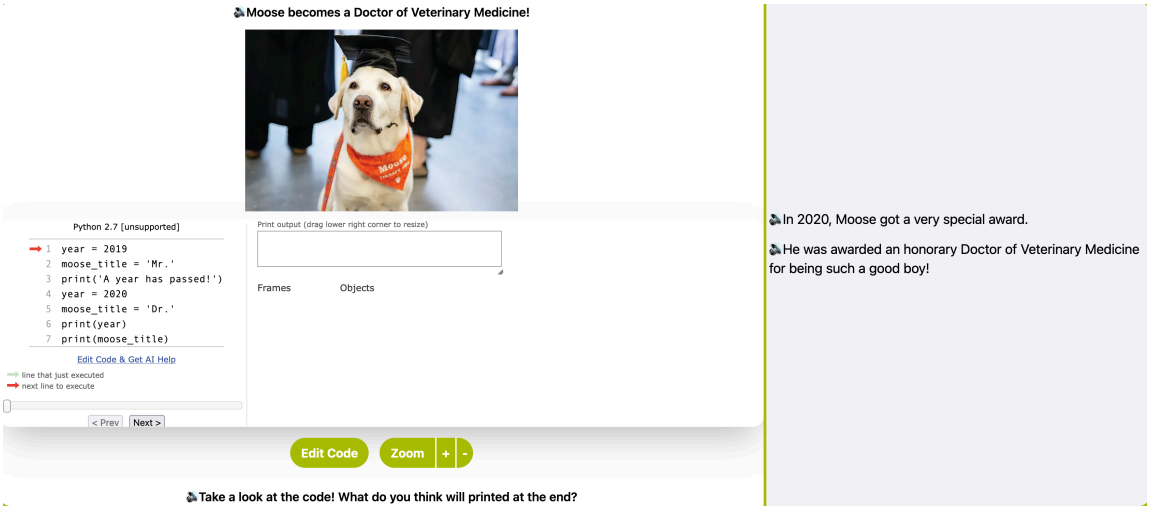


Figure 4.3: A page in the *Life of Moose* book using Python Tutor to showcase important events in Moose's life.

Figure 4.3 presents an example of one of the pages in the *Life of Moose* book. There are sections that describe moments of Moose's life that are followed by question sets using Python Tutor for students to answer, which can be seen in Table 4.3.

Table 4.3: Misconceptions targeted in the *Life of Moose* book.

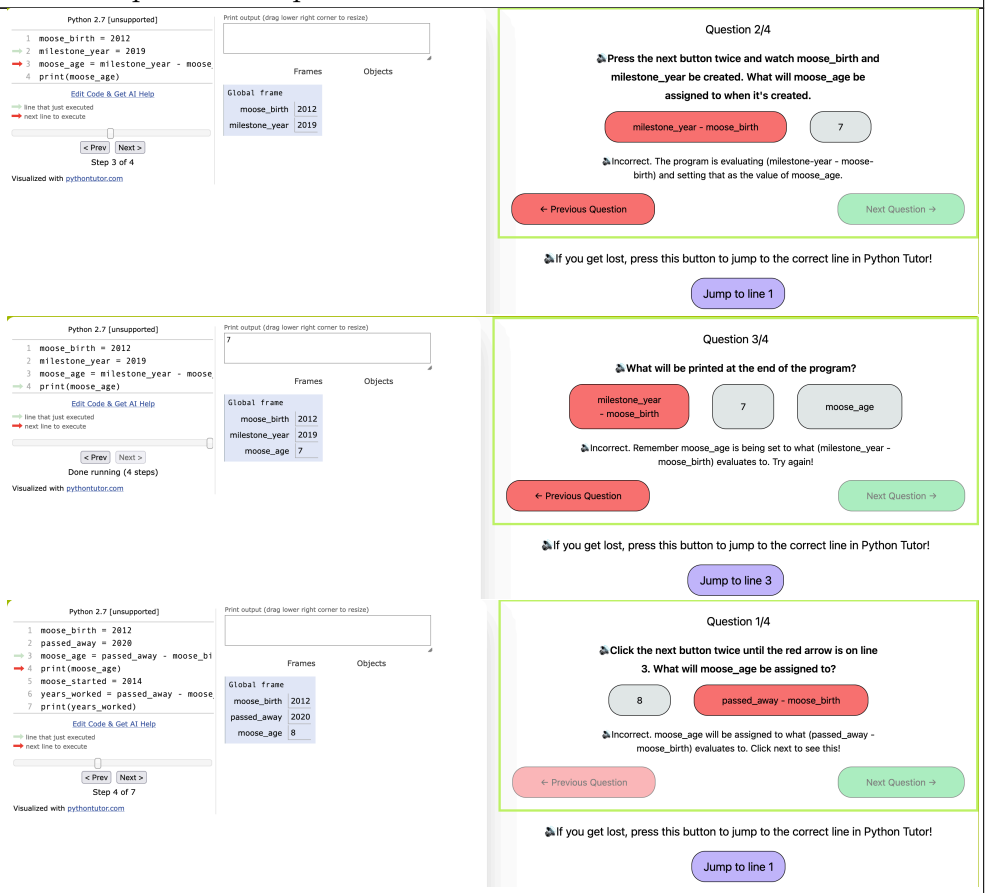
Misconception	Misconception Example
Assigning an expression to a variable instead of a value.	

Table 4.3: List of misconceptions targeted in the Life of Moose book.

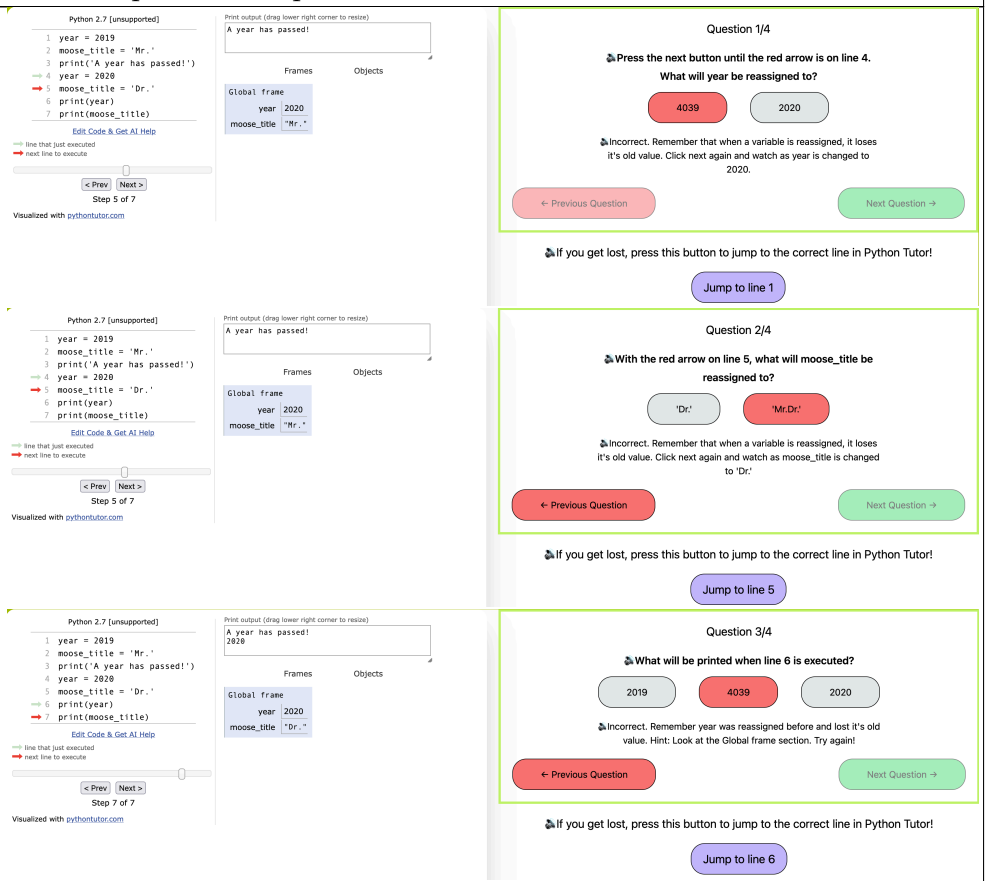
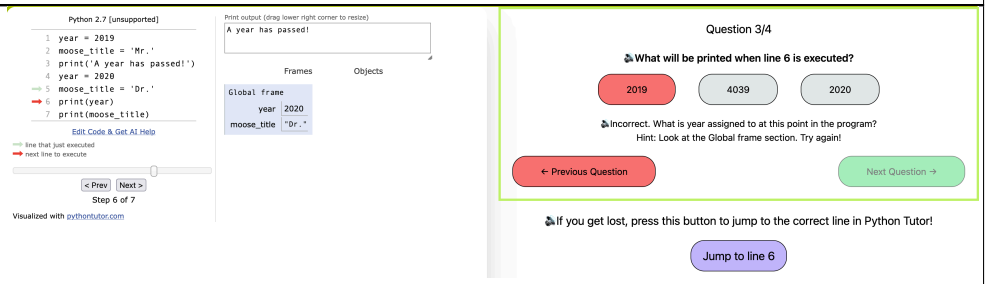
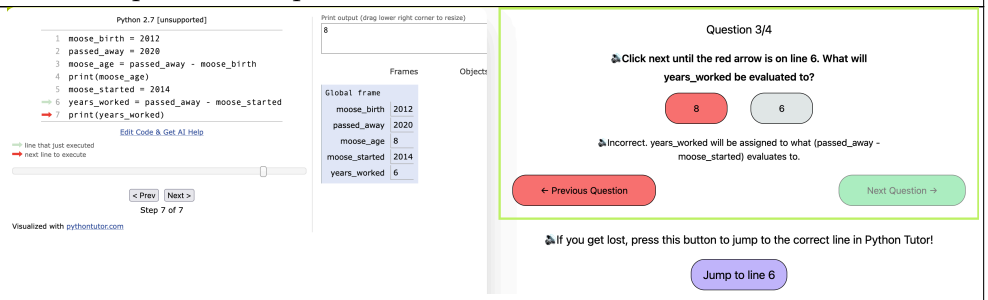
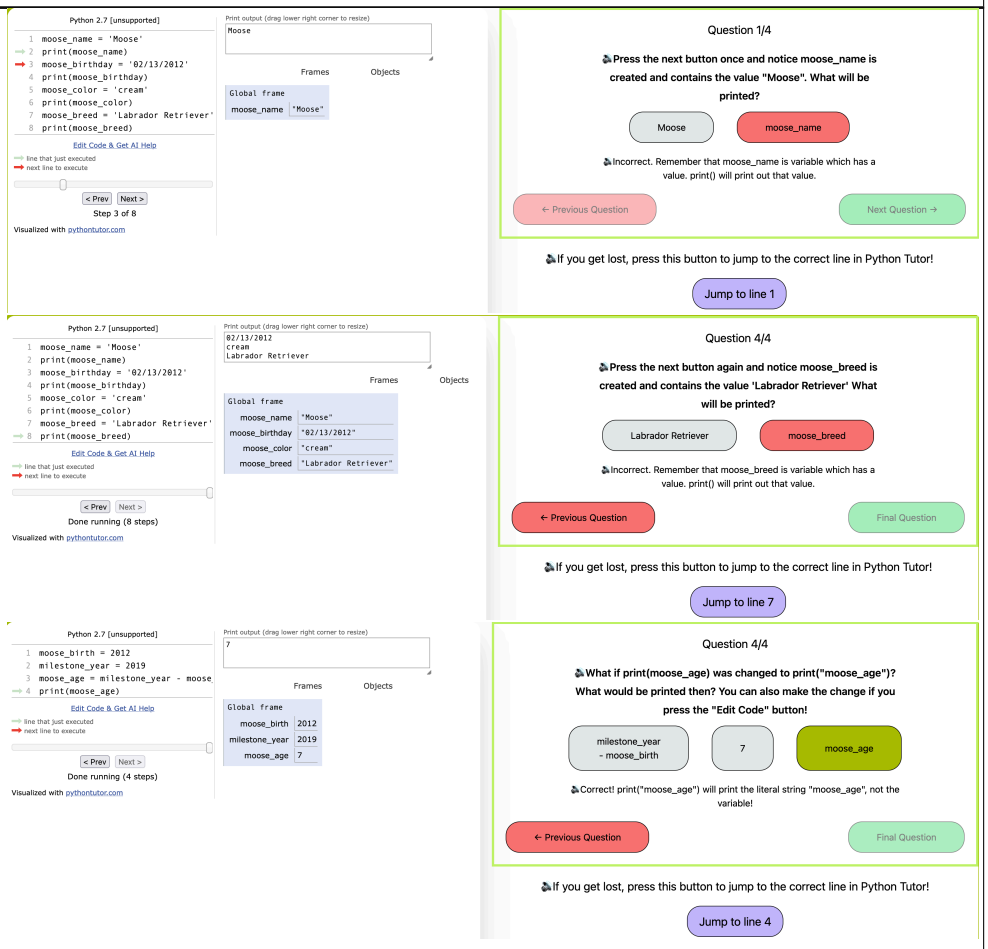
Misconception	Misconception Example
Summation of a variable's value when it's reassigned.	
Using the first or previously assigned value of a variable.	

Table 4.3: List of misconceptions targeted in the Life of Moose book.

Misconception	Misconception Example
Code doesn't execute sequentially.	
Using the variable's symbolic name instead of its value.	

4.2 Conditionals Books

These books are made to introduce children to the various aspects of conditional control flow in programming. Misconceptions related to conditionals are fewer than those from variables. For this reason, there are fewer examples of questions targeting misconceptions in the following books than in the variable books. In fact, the *Conditional Operators* and *Flowcharts* books don't address any misconceptions. Regardless, these books are still necessary for the children's fundamental understanding of conditionals and how they can be used to control the flow of execution in a program. This is because foundational computer science structures, such as if statements and loops, often use conditional and logical operators to control the flow of a program's execution.

4.2.1 Conditional Operators

The Conditional Operators book teaches children about the three primary conditional operators:

1. Equal to (==)
2. Less than (<)
3. Greater than (>)

Although this book doesn't contain any misconceptions from the previous literature, it is vital for the children's future understanding of if-statements and loops since

both of these structures often use conditional operators to make decisions.

This book uses three of the current Virginia Tech therapy dogs (Derek, Wagner, and Josie) and their attributes to teach children about conditional operators and how they evaluate to true or false. Figure 4.4 shows a page from the *Conditional Operators* book that uses the fur color to demonstrate the "equal to (==)" operator and how it will evaluate to false when comparing the fur color of two dogs with different colors.

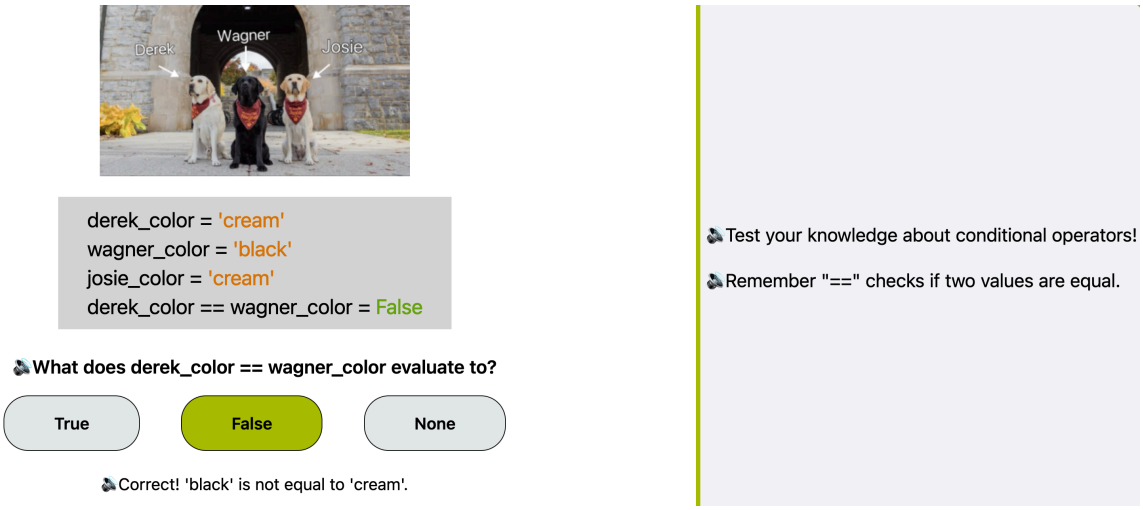


Figure 4.4: A page from the *Conditional Operators* book that uses Virginia Tech's therapy dogs to teach children the "equal to" operator.

Python Tutor is also used to show real code examples of conditional operators. Figure 4.5 shows the usage of the "equal to (==)" operator in Python to compare the values of two variables. In this example, the variable's values are the fur colors of the therapy dogs, and two print statements are used to showcase when the "equal to" operator evaluates to true and false.

Figure 4.5: A page from the *Conditional Operators* book that uses Python Tutor to demonstrate conditional operators in code.

4.2.2 Logical Operators

After learning about conditional operators, the *Logical Operators* book teaches children how to combine several conditions using logical operators. The three logical operators taught in this book are:

1. **AND** operator
2. **OR** operator
3. **NOT** operator

Like the *Conditional Operators* book, this book features Virginia Tech's three therapy dogs to help engage students and present the content in a familiar and accessible way compared to more abstract examples. Figure 4.6 shows an example from the book where the therapy dogs are used to teach children about the OR operator. The example uses the dogs' fur colors and the OR operator to determine if at least one of the dogs has a specific fur color.

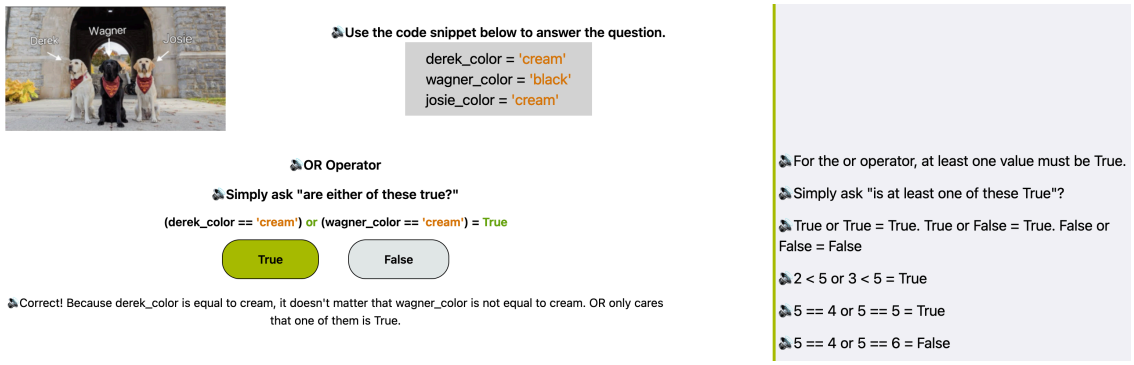
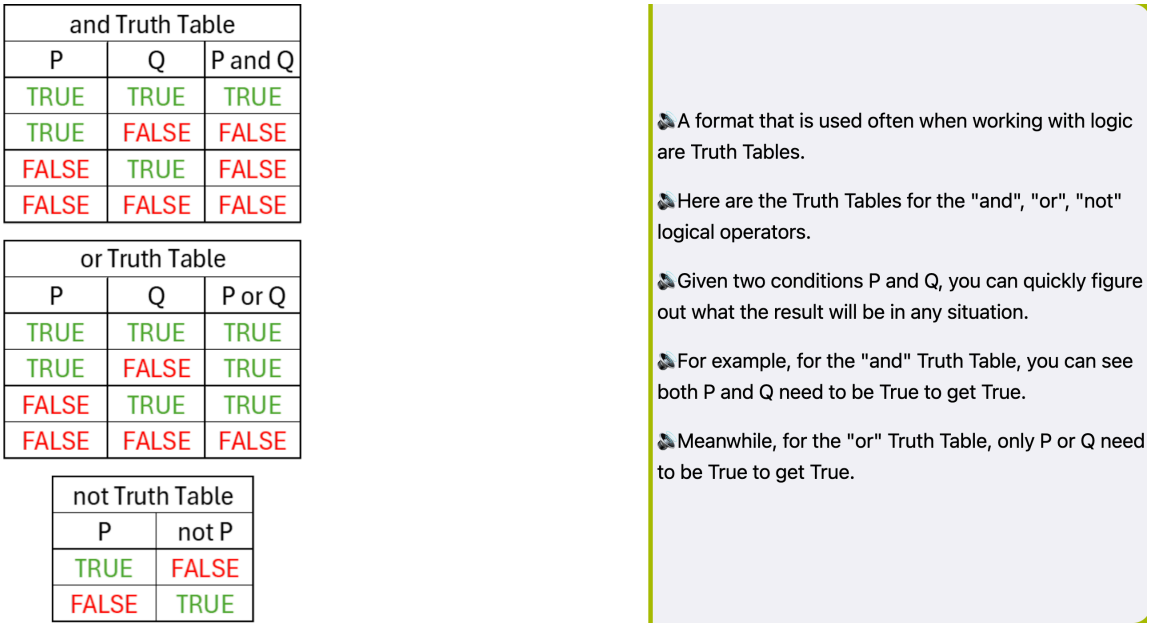


Figure 4.6: A page from the *Logical Operators* book that uses Virginia Tech's therapy dogs and their hair color to teach children the OR operator.

Python Tutor is used as well, with Figure 4.7 showing how conditional and logical operators can be combined to solve complex logical expressions. Again, variables are used whose values are the dogs' fur colors to demonstrate the AND, OR, and NOT operators.

Figure 4.7: A page in the *Logical Operators* book that uses Python Tutor to demonstrate conditional and logical operators in code.

The children were also introduced to truth tables in this book. Figure 4.8 shows the page in the book that teaches them Truth tables. They are taught the truth tables for the AND, OR, and NOT operators.



and Truth Table		
P	Q	P and Q
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

or Truth Table		
P	Q	P or Q
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

not Truth Table	
P	not P
TRUE	FALSE
FALSE	TRUE

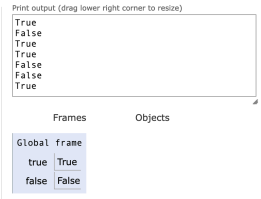
Figure 4.8: A page in the *Logical Operators* book that teaches children how to read and use truth tables.

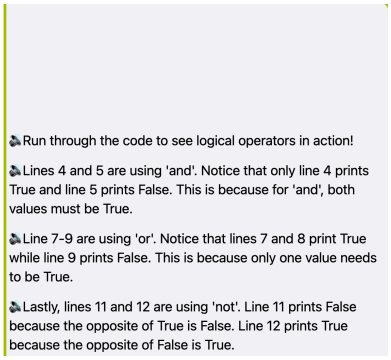
The misconception this book targets is the belief that the OR operator acts like the XOR operator. Meaning, students believe that when two conditions are true and are compared with the OR operator, the result is false. This misconception is clarified through both the truth tables and a Python Tutor example, as can be seen in Table 4.4.

Misconception	Misconception Type	Misconception Example	Discussion

OR operator doesn't evaluate to True when both values are True. (Like XOR)	**Logical Operators** - the student misunder-stands how the OR operator works.	x = True
		y = True
		if x or y:
		  print("True")
		else:
		  print("False")
		
		Student Answer: "False"
		Correct Answer: "True"
			The student thinks the OR logical operator evaluates to false when both conditions are True.
			
			Grover and Basu [6] found that 6.5% of students held this misconception.

Book Example





and Truth Table		
P	Q	P and Q
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

or Truth Table		
P	Q	P or Q
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

not Truth Table	
P	not P
TRUE	FALSE
FALSE	TRUE

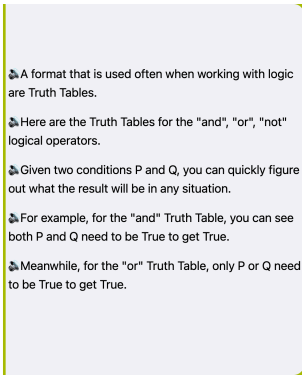


Table 4.4: Or operator misconception targeted in the *Logical Operators* book.

4.2.3 Flowcharts

The *Flowcharts* book introduces children to flowcharts and is intended to familiarize them with the concept of program control flow before learning about if-statements. To make the concept more engaging, the book utilizes a food menu shown in Figure 4.9. The food menu features foods children enjoy and their prices, which are incorporated in flowcharts to determine whether a food item can be bought.

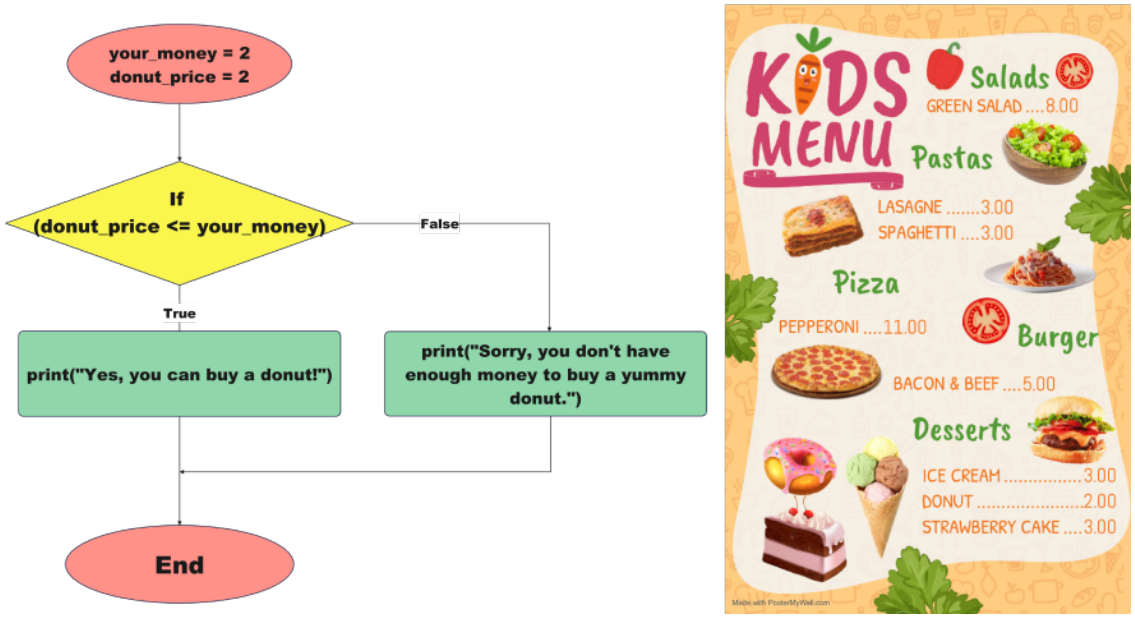
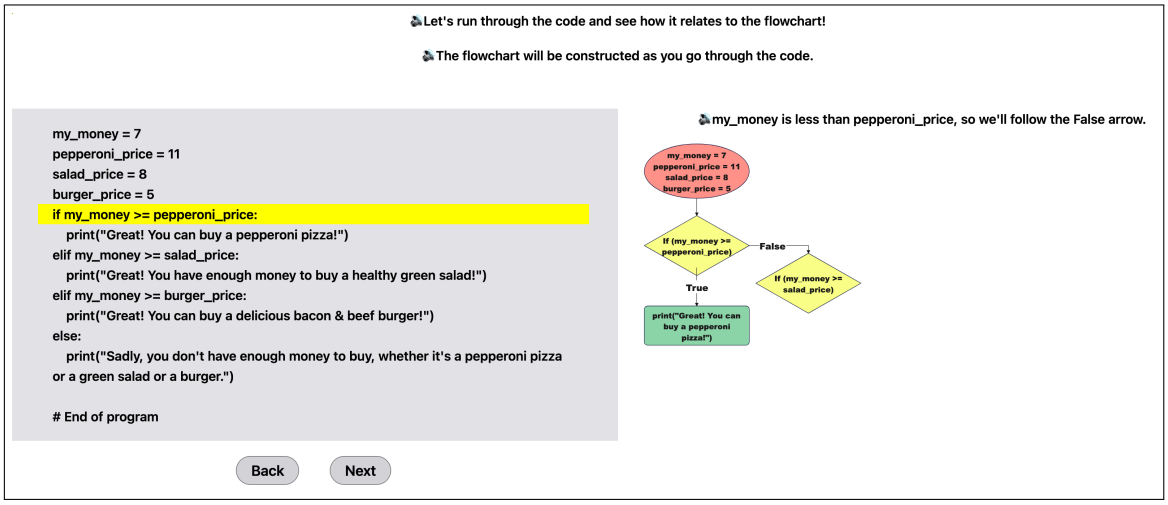
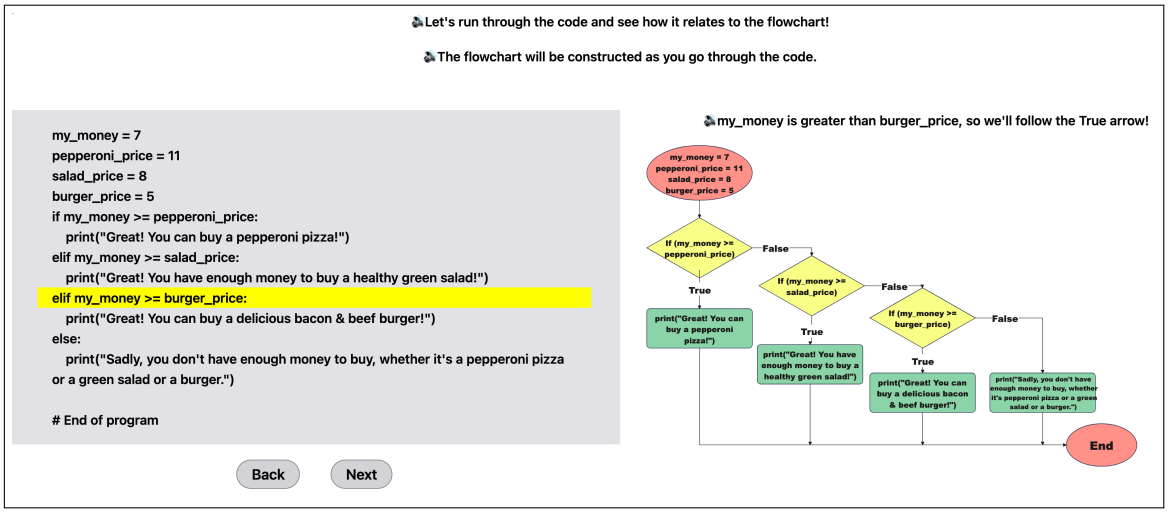


Figure 4.9: The food menu with prices used in the *Flowcharts* book to teach children flowcharts and how conditions can be used to control what code is executed.

The book also includes question sets and code snippets that build a flowchart as the "Next" button is pressed. Figure 4.10 shows an example of a code snippet and how a flowchart is built as the student progresses through the code. Sub-figure 4.10a shows the flowchart as it is partially built, and Sub-figure 4.10b shows the flowchart once it is fully built after the student has advanced through the code.



(a) A partially built flowchart is displayed as the student begins executing the code.



(b) The full flowchart is displayed once the student reaches the end of the if-else block.

Figure 4.10: A flowchart in the *Flowcharts* book is built as the "Next" Button is pressed.

This book does not directly target any misconceptions, but it teaches students that programs can be controlled. This knowledge is fundamental for the final conditional book where students combine what they've learned so far to learn about

if-statements.

4.2.4 If Statements

The *If Statements* book combines everything the students learned in the *Conditional Operators*, *Logical Operators*, and *Flowcharts* books to teach them how to use `if statements` to control a program's flow of execution. While the *Flowcharts* book uses `if statements` in its flowchart examples, this book formally introduces the concept and uses code examples to show its usage in text-based programming. Like previous books, this book uses Virginia Tech's therapy dogs and includes interactive examples using both Python Tutor and multiple-choice questions. Figure 4.11 shows a page from the *If Statements* book that uses the Virginia Tech therapy Derek to ask a question about an if-else block.

The common `if statements` misconception targeted in this book is students believing that the program will terminate immediately if a condition is false and there is no else branch. More details about this misconception can be referenced in Table 4.5.

🔊**What is printed in the program?**

```
derek_is_happy = True
if derek_is_happy:
    print("True")
else:
    print("False")
```

🔊**Notice that only the print("True") is part of the If-statement. The print("False") is part of the else.**

True False	**True**	**False**	**Nothing will be printed.**

🔊Correct! derek_is_happy is True so everything indented under the If-statement will be executed and the else will be skipped.

Figure 4.11: A page from the *If Statements* book using Virginia Tech's therapy dog Derek to teach students about the else block.

<table>
<tr><th>Misconception</th><th>Misconception Type</th><th>Misconception Example</th><th>Discussion</th></tr>
<tr><td>False condition ends the program if there's no else.</td><td>Condition - the student believes a false condition stops the entire program

Sequence - the student believes the sequential execution of code stops on a false condition.</td><td>x = 1
if x == 2:
  print(“True”)

print(“False”)

Student Answer: Nothing is printed.
Correct Answer: “False”</td><td>The student thinks that when the if-condition fails with a False condition, the program will end. The book example contains an if-statement that will evaluate to false without an else block.

Swidan et al. [15] found that 7.4% of students hold this misconception.</td></tr>
<tr><td colspan="4">Book Example
 

What is printed in the program below?
anniversary = 148
if anniversary == 200:
print("True")

print("False")

True
False
True
False
Nothing will be printed.
Incorrect. A program will continue to execute even if a If-statement's condition is False. Also, the final print is not indented, so it's not part of the If-statement. Try again!

The yellow highlight has been removed from the If-statements, so it's up to you now to notice the indents!
Try out the exercise to the left!</td></tr>
</table>

Table 4.5: If Statement misconception targeted in the *If Statements* book.

4.3 Loops Books

These books introduce students to loops in programming. Due to loops being an advanced concept, there are many misconceptions related to them. To make the learning more engaging for young students, these books use a farm theme and incorporate activities like riding a pony and feeding goats to teach children the behavior of loops.

This topic was split into two books to teach the two most common types of loops in programming: for and while loops.

4.3.1 For Loops

The *For Loops* book introduces children to the concept of loops in general, focusing specifically on `for loops`. Students build upon their previous knowledge about programming control flow to learn how to execute pieces of code multiple times using the "counting loop" (`for loops`).

As previously mentioned, this book uses farm-themed examples such as feeding goats a specific number of cups of food and riding three laps on a pony. The book also uses flowcharts and dynamically highlights pieces of the flowchart depending on the step they are on in the loop. Figure 4.12 shows an example from the book where the specific part of the flowchart is highlighted depending on the current step and iteration of the `for loops`.

This book targets several loop misconceptions related to `for loops`, which can be

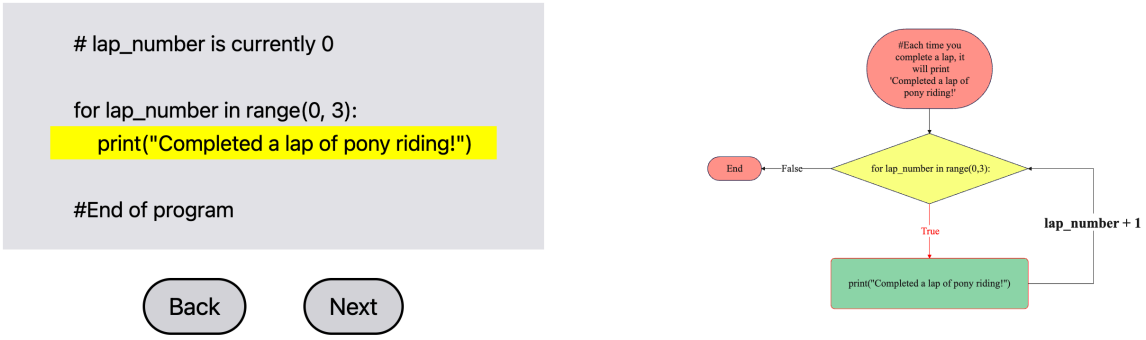


Figure 4.12: An example of a `for loop` in the *For Loops* book that uses a flowchart to demonstrate the flow of execution.

seen in Table 4.6. Common misconceptions include believing code adjacent to the loop executes as part of the loop, misunderstanding that loops execute sequentially, and adding or missing iterations of a loop.

Table 4.6: Misconceptions targeted in the *For Loops* book.

Misconception	Misconception Type	Misconception Example	Discussion
Adjacent code executes when the loop is running.	**Loop Scope** - the student believes code directly after the loop executes with the loop.	number = 0 for x in range(5): number = number + 1 print(number) Student Answer: "1 2 3 4 5" is printed. Correct Answer: "5" is printed.	In the book example, the student doesn't recognize that the print statement is outside of the loop control structure and will only execute once the loop is finished. Swidan et al. [15] observed that 33.3% of students held this misconception.

Book Example

```
1 goats_fed = 0
2 for x in range(0, 3):
3   goats_fed = goats_fed + 1
4 print(goats_fed)
```

What will be printed when this code executes?

1 2 3 | 3

Incorrect. Notice how print(goats_fed) is not within the loop. Try again!

Table 4.6: Misconceptions targeted in the Variables Data Types book..

Misconception	Misconception Type	Misconception Example	Discussion
The code inside a loop isn't executed sequentially but is instead grouped.	**Code Sequence** - the student doesn't understand that the code executes sequentially in the loop.	for x in range(3): print("Start") print("Stop") Student Answer: "Start Start Start Stop Stop Stop Done" Correct Answer: "Start Stop Start Stop Start Stop Done"	The student thinks each line of code in a loop is grouped instead of executing sequentially. The book example uses the laps of a race to target this. Grover and Basu [6] observed that 8% of students held this misconception.

Book Example

```
1 print('Go!')
2 for lap in range(1, 4):
3     print('Starting lap ' + str(lap) + '!')
4     print('Ending lap ' + str(lap) + '!')
5 print('Race finished!')
```

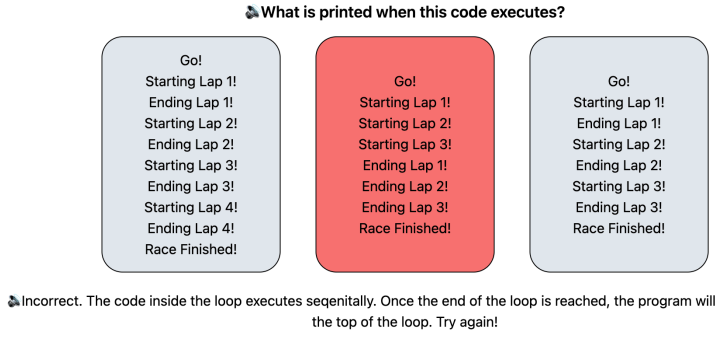


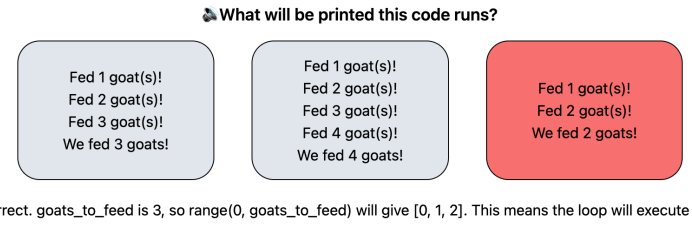
Table 4.6: Misconceptions targeted in the Variables Data Types book..

Misconception	Misconception Type	Misconception Example	Discussion

The student either misses or adds an extra iteration of the loop when using a preassigned variable.	**Loops** - student incorrectly determines the number of iterations a loop will have.	times = 3
		count = 0
		for x in range(times)
		  count = count + 2
		  print(count)
		print("Last:" + str(count))
		
		Student Answer: "2 4 Last: 4" or "2 4 8 Last: 8"
		Correct Answer: "2 4 6 Last: 6"
			The student doesn't understand how many times a loop will execute given a preassigned variable. The book example has students determine how many times a goat will be fed.
			
			Grover and Basu [6] found that 7.8% of students added a count and 3.9% of students missed a count.

Book Example

```
1  goats_to_feed = 3
2  fed_goats = 0
3  for cup_number in range(0, goats_to_feed):
4    fed_goats = fed_goats + 1
5    print("Fed " + str(fed_goats) + " goat(s)!")
6
7  print("We fed " + str(fed_goats) + " goats!")
```



4.3.2 While Loops

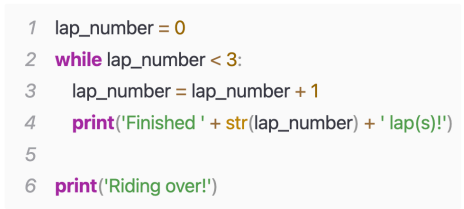
The *While Loops* book introduces children to `while loops` and continues with the farm theme from the *For Loops* book. The examples in this book are similar to those in the *For Loops* book, but they now use `while loops`. This book also uses more of their prior knowledge about conditional control flow, as `while loops` rely directly on conditions to determine whether to continue executing or not.

This book targets one misconception where children believe a `while loop` will stop executing as soon as the condition becomes false. More information on this misconception can be seen in Table 4.7.

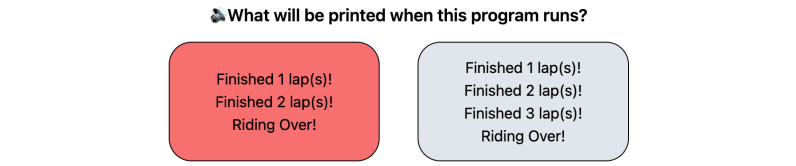
Misconception	Misconception Type	Misconception Example	Discussion

A while loop terminates as soon as the condition becomes false.	**Code Sequence** - the student doesn't understand that code is sequential and that when a loop iteration starts, the entire loop will be executed.	x = 1
		while x < 3:
		  x = x + 1
		  print(x)
		
		Student Answer: Only "2" is printed.
		Correct Answer: "2 3" is printed.
			While the student understands that when the condition becomes false, the loop will terminate, they think the termination will happen instantly. The book example has students determine the number of laps that will be done.
			
			10.5% of students hold this misconception [15].

Book Example



```
1 lap_number = 0
2 while lap_number < 3:
3   lap_number = lap_number + 1
4   print('Finished ' + str(lap_number) + ' lap(s)!')
5
6 print('Riding over!')
```



Incorrect. Remember the loop will continue to it's current execution even if the condition it checks becomes false inside the loop.

Table 4.7: While loop misconception targeted in the *While Loops* book.

Chapter 5

Results

This chapter shows the results of the survey and examination. Due to time constraints, only the variables books (*Variables* and *Variable Data Types*) were used in a classroom environment. These books, however, make up the majority of all misconceptions targeted in this study.

5.1 Survey

A survey was conducted with 6th and 7th-grade students to get the general demographics of the students, their prior programming experience, and their opinions on the *Variables* and *Variable Data Types* books. The survey students completed can be referenced in Appendix B. The survey results also help test our first hypothesis that the books provided an enjoyable means for children to learn about computer science and increase their perceived knowledge of the learned concept. The survey received 97 responses from children aged between 11 to 14 years old.

5.1.1 Previous Programming Experience

Students were asked the following question regarding their prior programming experience: *How would you rate your previous experience with programming?*. Most students stated they have had minimal previous programming experience. Figure 5.1 shows the distribution of students' self-rated prior programming experience, with the plurality of students ranking themselves as two. Additionally, the median answer to this question is two, which indicates that most children had little to no programming experience. This lack of prior programming experience is significant to our study, as it highlights the need for foundational programming instruction in elementary and middle schools that CodeKids can provide.

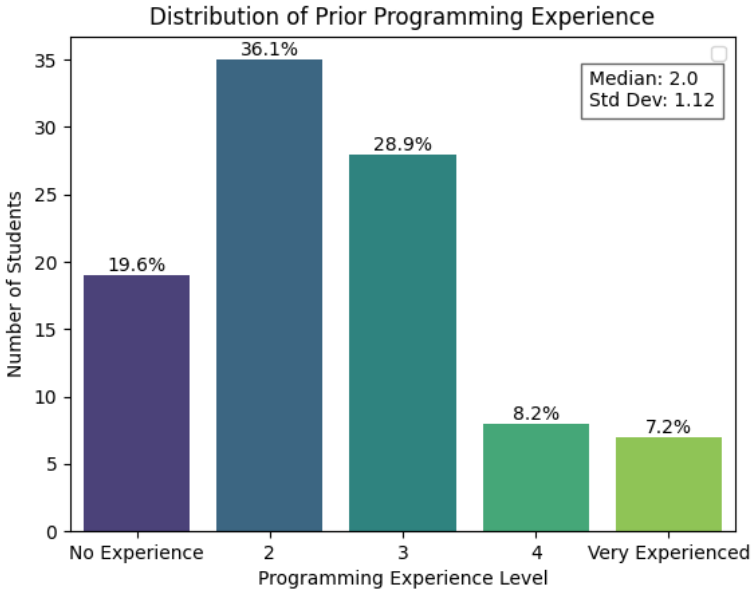


Figure 5.1: A bar chart showing the distribution of students' prior programming experience. The median experience level is two, indicating that most students have little to no experience.

When asked about previous programming languages students have used, 63.9% of the responses indicated no prior programming language use. Of the children that did have programming experience, Scratch (15.5%), Python (15.5%), Java (10.3%), and Blockly (4.1%) were the most commonly reported. Figure 5.2 shows the number of students who have used different programming languages. It is important to note that children could select multiple languages, resulting in a total of 117 responses for this question. Additionally, some students filled in blank or invalid responses in the "Other" category, which were labeled as N/A. This figure also indicates the significant lack of programming experience for most students, reinforcing the need for platforms like CodeKids to introduce foundational programming concepts.

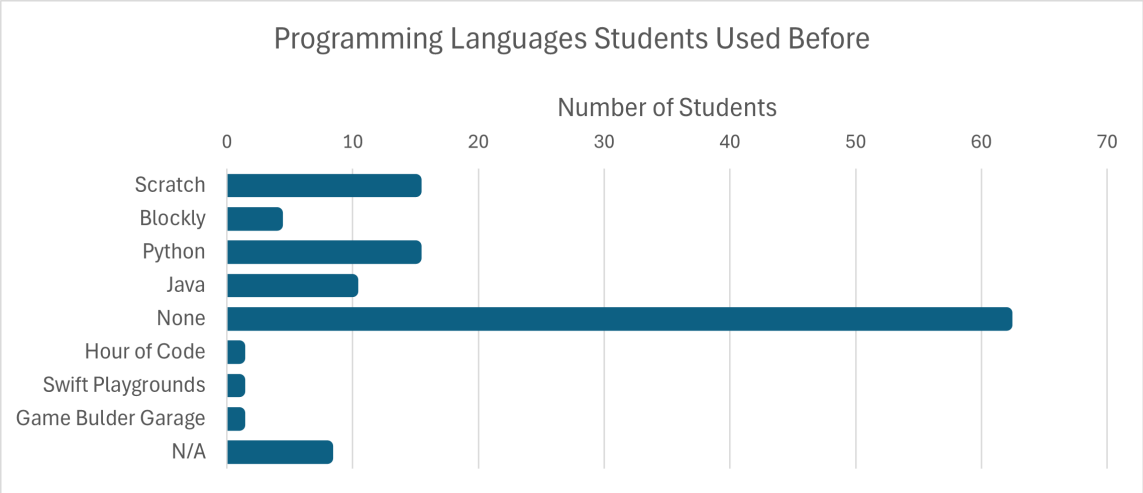


Figure 5.2: A chart showing which programming languages and environments students have used in the past. The majority indicated no prior experience. N/A responses include blank answers and non-programming language responses.

Some other programming environments were also mentioned by students. Hour of Code is a set of activities from Code.org that encourages children to code. Game

Builder Garage is an application for the Nintendo Switch that lets children make video games and learn the basics of game design using visual programming. Lastly, Swift Playgrounds is an app for the iPad and Mac that helps people learn how to code and build apps using Swift.

Additionally, students were asked *Are you interested in programming?* and ranked their interest on a scale of 1 to 5. 62.8% of students indicated they are at least somewhat interested in programming, and 37.2% did not show much interest. This suggests that despite the majority of students having little programming experience, the majority of students are still open to learning how to program. Figure 5.3 shows the distribution of students and their self-ranked programming interest. The plurality of students indicated being somewhat interested in programming, and 21.6% were "very interested". However, the number of students showing little interest is still significant and highlights potential challenges for the broader implementation of computer science education in elementary and middle schools, as well as the challenges faced by platforms like CodeKids to foster interest in children.

Lastly, students were asked the following question: *Have you used variables in programming before reading these books?*. 74.2% of students reported not having any experience with variables in programming before using the books, as shown in Figure 5.4. This means that for the majority of students, these books were the first exposure they had to variables in computer science. Again, this finding underscores the importance of resources like CodeKids in broadening computer science education in elementary and middle school classrooms and teaching computational concepts to children.

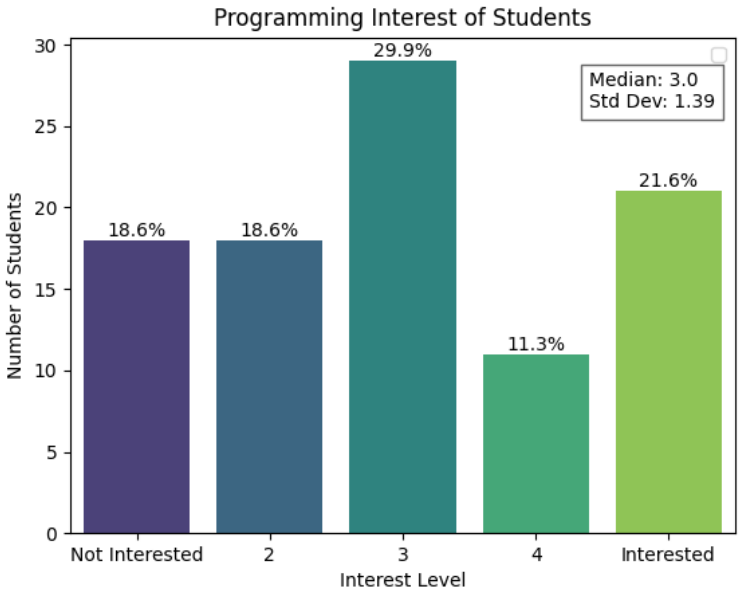


Figure 5.3: A bar chart showing the students' interest in programming. The median response is three, indicating that the plurality of students have a moderate interest in programming.

5.1.2 Python Tutor

Python Tutor is a tool utilized in the books to allow children to step through code line by line and provide a visual representation of how a program executes and manipulates variables. This tool can be specifically useful in helping students understand the dynamic behavior of code. Students were asked the following question: *How useful was Python Tutor to learn about variables?*. The plurality of students (33%) thought it was somewhat useful. Additionally, 38.1% of students leaned towards more useful and 28.9% leaned more towards less useful. Overall, the scale of students who thought Python Tutor wasn't useful to students who thought it was

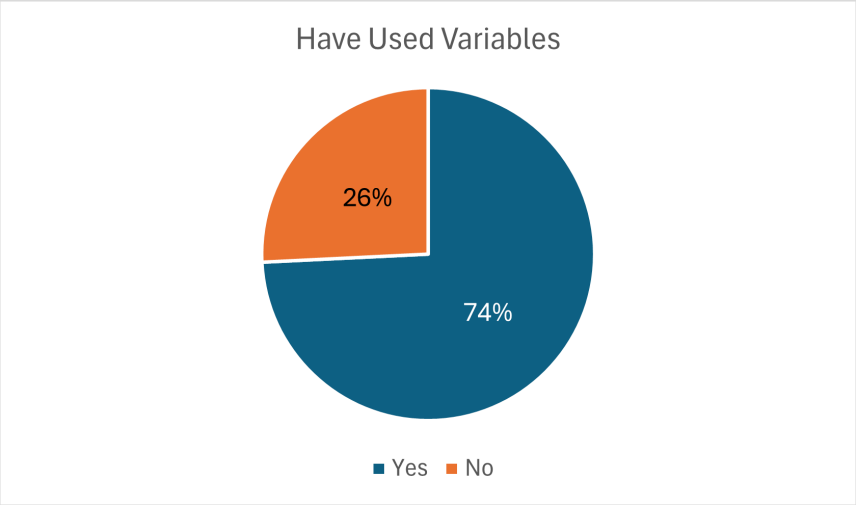


Figure 5.4: A pie chart showing the proportion of students who have used variables in programming. 74% of students have not used variables before.

very useful is distributed fairly evenly, which can be seen in Figure 5.5.

This mixed feedback indicates varying levels of engagement and effectiveness that students experienced with Python Tutor. While a substantial number of students did find the tool helpful, a significant number did not, which suggests the tool is not equally effective for all learners. This puts emphasis on the need to consider diverse learning preferences and explore additional tools to better support student learning.

5.1.3 Opinions on Books

Students were asked the following question regarding their perceived difficulty of the books: *How difficult were the books?*. The plurality of students (40.2%) thought the books were somewhat difficult. A smaller proportion of students (13.4%) leaned more towards the books being difficult, and the rest (46.4%) leaned more towards the

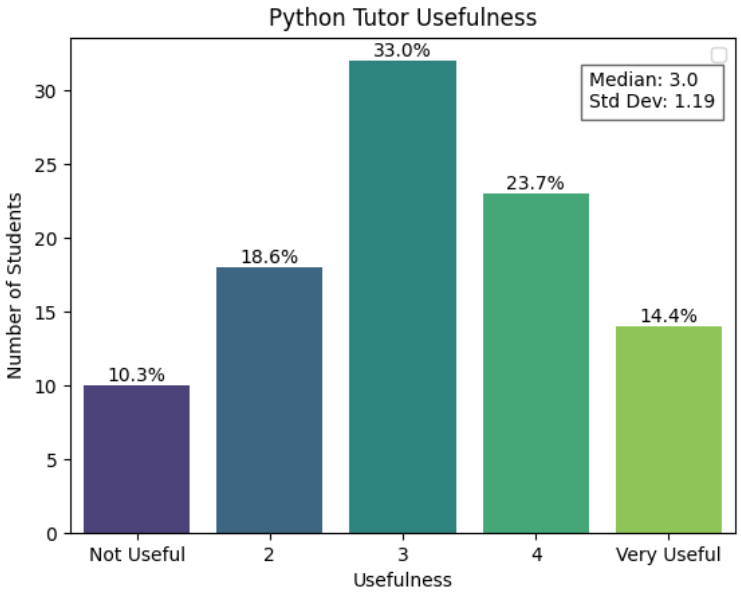


Figure 5.5: A bar chart showing the distribution of students' perceptions of Python Tutor's usefulness in the books. The median response is three, indicating that the plurality of students thought Python Tutor was moderately useful.

books being easy. Figure 5.6 shows this distribution of student perceptions regarding the difficulty of the books.

This finding suggests the books strike a good balance in terms of difficulty for most students. This aligns with the goal of the books to be moderately challenging but not so challenging that students feel less motivated to learn computer science or so easy that children become bored using the books. However, some students still found the books to be too difficult, and a significant number thought the books were too easy. This feedback suggests the need for more adaptive content relative to the students' current knowledge.

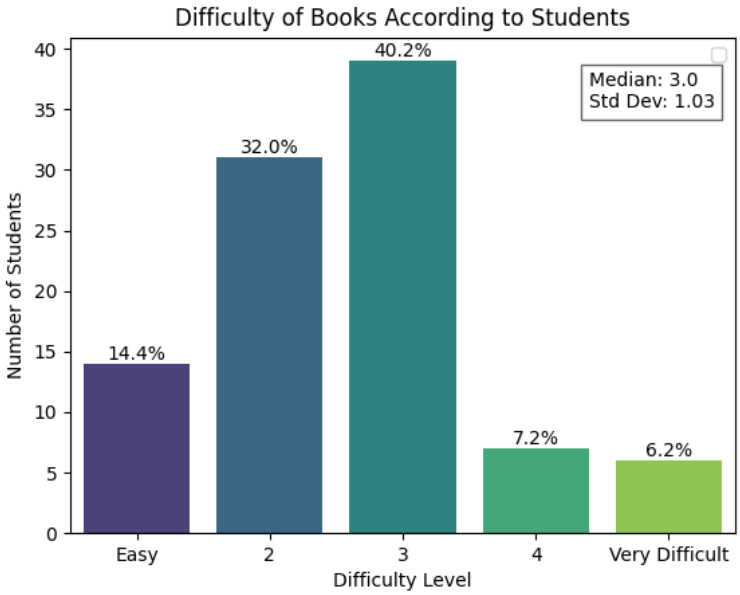


Figure 5.6: A bar chart showing the distribution of how difficult the students thought the *Variables* and *Variables Data Types* books were. The median response was three, indicating the plurality of students thought the books were moderately difficult.

For the examples and activities in the books, students were asked the following question: *Did you like the examples/activities the books used?*. Figure 5.7 shows that 82.5% of the students stated they liked the activities. Some students went into further detail explaining why they liked the activities. One student stated "I loved them and they helped me better understand what the lesson was trying to tell me". Another stated how the activities helped them because the content is confusing on its own "I liked them a little bit because is was a bit confusing".

However, 17.5% of students didn't like the examples and activities, with many describing them as boring or hard. One student wrote "Coding is super duper hard and I don't like it", while another student expressed frustration with comprehension:

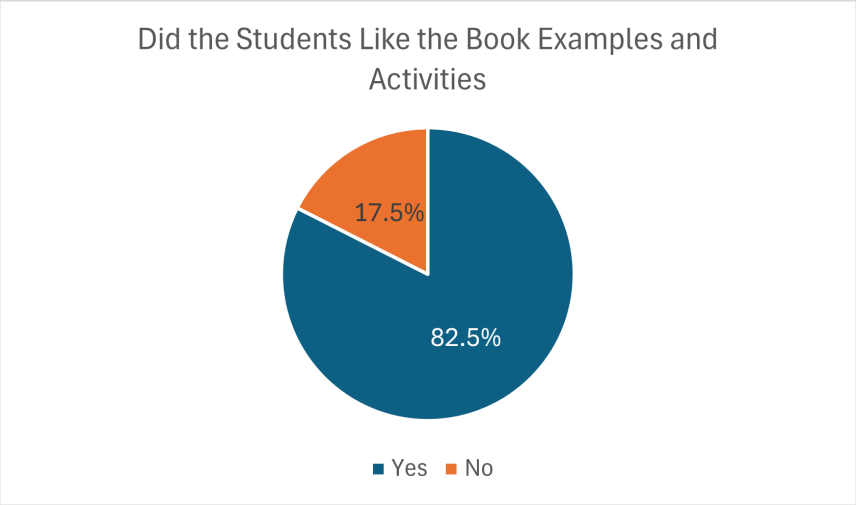


Figure 5.7: A pie chart showing the proportion of students who liked the examples and activities in the *Variables* and *Variable Data Types* books. 82.5% of students of students liked the examples and activities in the books.

"It wasn't real easy for me to comprehend in my brain". These comments highlight the challenges some students face, which may stem from varying levels of prior experience, interest in programming, and learning preferences.

5.1.4 Perceived Knowledge

Students' perceived knowledge of variables was assessed before and after using the CodeKids variables books. They were asked the following question: *How would you rate your knowledge of variables before and after reading the books?*. Before the books, 77.3% stated they had little or no knowledge of variables, while the remaining students believed they had some or a lot of knowledge. After using the books 69.1% of students believed they had at least some knowledge of variables. However, 21.6%

of students still believed they had little knowledge, and 9.2% still believed they had no knowledge at all. Figure 5.8 shows the proportion of students who categorized themselves in each knowledge rating before and after using the *Variables* and *Variable Data Types* books.

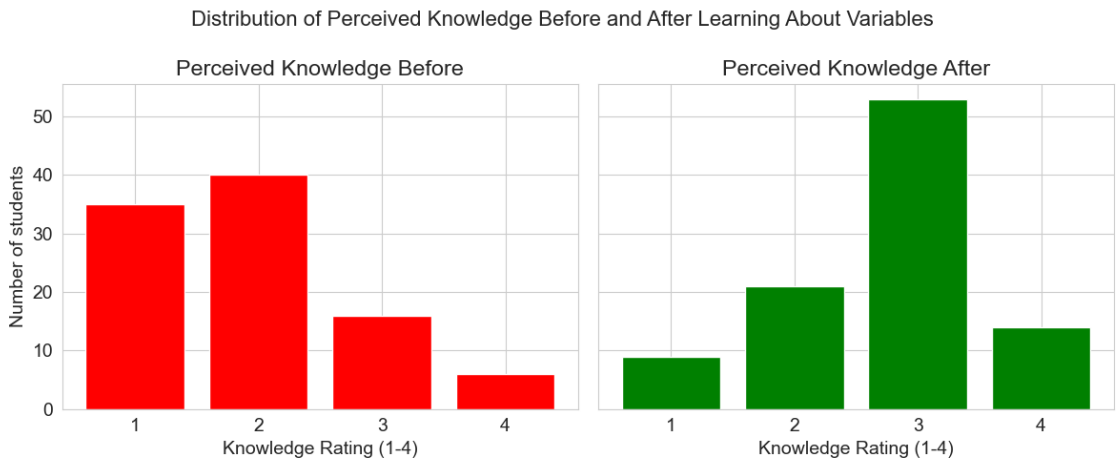


Figure 5.8: Distribution of students' self-reported perceived knowledge of variables before and after using the CodeKids variables books. (1 = No Knowledge, 2 = Little Knowledge, 3 = Some Knowledge, 4 = A lot of knowledge)

To evaluate the statistical significance of the students' change in perceived knowledge, the difference between their perceived prior knowledge before and after using the books was calculated for each student. Since the data is ordinal and the observations are paired, a Wilcoxon Signed-Rank test was used to determine significance. The hypotheses were:

- Null hypothesis H_0: The median difference in perceived knowledge is zero.
- Alternative hypothesis H_A: The median difference in perceived knowledge is greater than zero (knowledge increased).

This test showed a statistically significant increase in perceived knowledge after using the books with a test statistic $W = 262$ and a p-value $p < 0.001$. This means there is strong evidence that the children's perceived self-knowledge of variables increased after using the books. Figure 5.9 shows a box plot of this shift on a broad scale, and Figure 5.10 shows the shift of perceived knowledge for individual students using a paired dot plot.

While this isn't an indication of actual knowledge, it demonstrates CodeKids effectiveness in improving students' confidence towards variables. However, the fact that a significant number of students still reported little to no knowledge of variables suggests that additional support and teaching strategies are needed to address the needs of individual learners. This observation is important for refining the books and ensuring they are effective for all students.

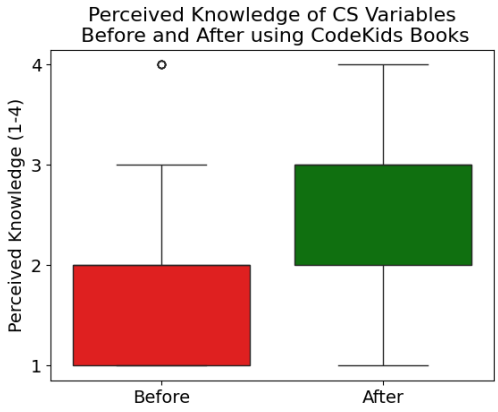


Figure 5.9: A box plot of students' perceived knowledge before and after using the CodeKids *Variables* and *Variable Data Types* books.

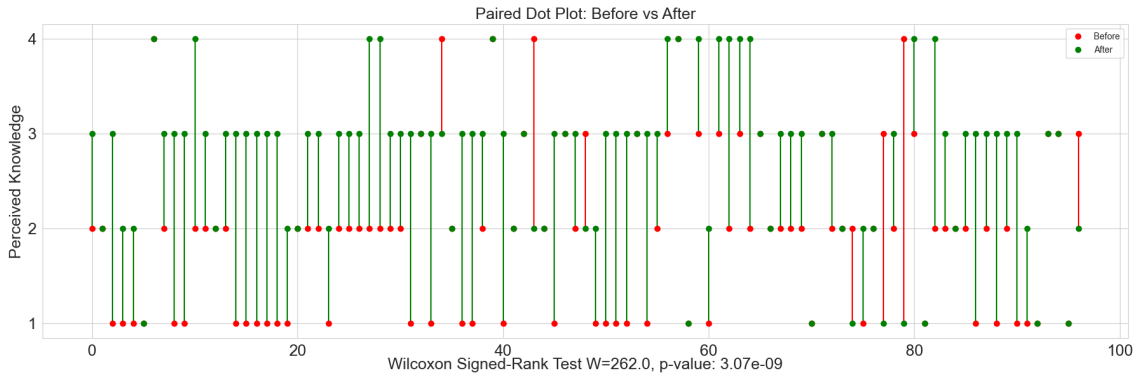


Figure 5.10: A paired dot plot of individual student's perceived knowledge before and after using the CodeKids *Variables* and *Variable Data Types* books. A green line represents an increase and a red line represents a decrease in perceived knowledge.

5.1.5 Correlation of Age

Several tests were done to test the correlation between the students' ages and different factors, including prior programming experience, programming interest, believing the books were difficult, liking the examples in the books, and prior use of variables. Spearman's rank correlation test was used to determine if there is a significant relationship between age groups and these factors since this test is suitable for ordinal data and measures the strength and direction of monotonic relationships. Ages ranged from 11-14, with the majority (50) of the children being 12 years old and the rest being 11 (26) and 13-14 (20) years old. For this reason, children were split into three groups for the analysis: 11, 12, and 13-14, which all have a population greater than five.

The following hypotheses were used for every test and its factor:

1. Null hypothesis H_0: There is no monotonic relationship between age and the

factor among 11-14 year old students ($\rho = 0$).

2. Alternative hypothesis H_A: There is a monotonic relationship between age and the factor among 11-14 year old students ($\rho \neq 0$).

For the previous use of variables, the Spearman correlation gave a weak positive correlation that was close to significance ($\rho = 0.18$, $p = 0.06$). This suggests that older students may have been slightly more likely to have used variables in the past, but the relationship is not statistically significant. However, with a p-value close to significance, it's possible a larger data set could show a stronger relationship. Figure 5.11 shows this relationship, and the proportions show a slight trend towards older students, particularly those older than 11, possibly having more experience with variables.

For the remaining correlation tests, all p-values were well above 0.05 with correlations close to zero. This indicates there is no evidence of a correlation between age and all the factors mentioned. Figure 5.12 shows several stacked bar charts of all these tests. From these charts, it is noticeable that the proportions between age groups are similar for all rankings.

These findings suggest age is not a strong predictor of students' prior programming experience, interest, or perceptions of the books. This implies the design of the books is broadly applicable across the 11-14 age range without needing significant age-specific adaptations. Instead, adaptations may be more necessary for the knowledge level of a particular student. Additionally, the fact that older students are not more likely to have prior programming experience has implications regarding meeting the

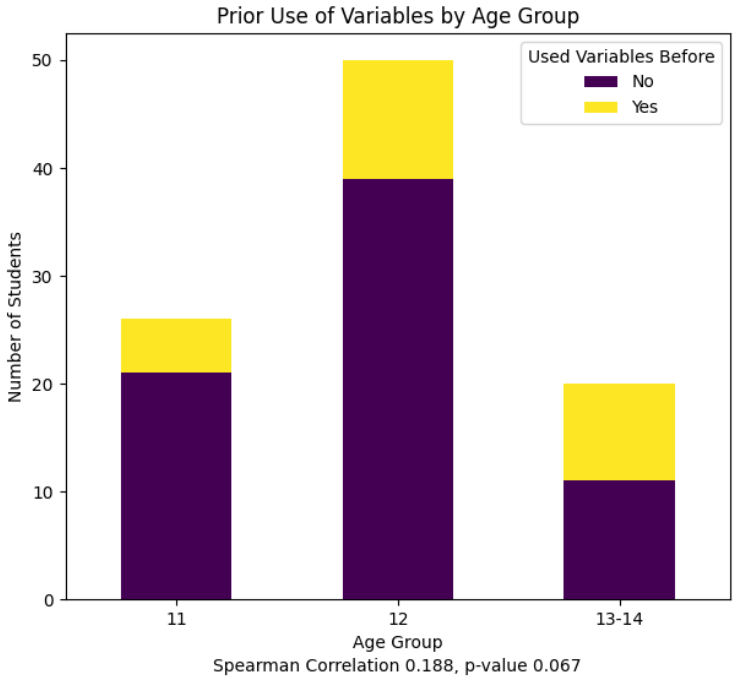


Figure 5.11: A stacked bar chart showing the relationship of age groups and whether students had used variables before using the *Variables* and *Variable Data Types* books. The Spearman correlation test indicated a slight trend of older students being more likely to have used variables that is close to significance.

computer science learning standards. The students in this grade level should have at least used variables by the fourth grade [16], and there is an expectation that older students would have more programming experience than younger students if the learning standards had been met. The implications of this will be discussed more in Sub-section 6.1.1. It is evident these standards have not been met, highlighting the need for more tools and platforms like CodeKids to teach computer science to K-8 students.

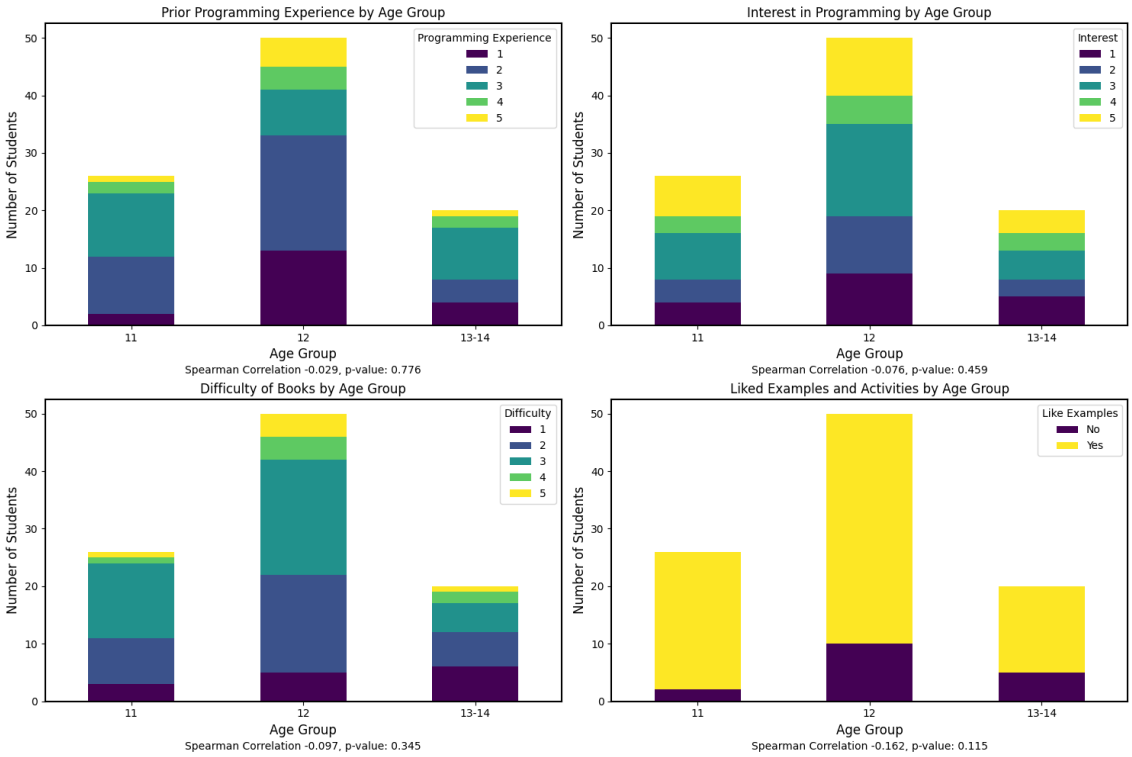


Figure 5.12: Stacked bar charts of all the Spearman correlation tests examining the relationship between the students' age and several factors. The factors include prior programming experience, interest in programming, perceived difficulty of the *Variables* and *Variable Data Types* books, and whether or not they liked the examples and activities in the books. For factors ranked 1-5, 1 denotes a lesser ranking (e.g. no programming experience or easy) and 5 denotes a higher ranking (e.g. very experienced or very difficult).

5.2 Examination

The examination was given to 6th and 7th-grade students after they had used the *Variables* and *Variable Data Types* books, and 115 students took the exam. The exam contained multiple-choice questions with incorrect answers that targeted misconceptions the books addressed. The exam can be referenced in Appendix A with red text indicating the misconception targeted and the incorrect answer depicting the misconception.

The proportion of students who chose the misconception answers was compared with the proportion of students who held the misconception in the previous studies listed in the literature review in Section 2.3. Additional misconceptions were also observed regarding the string data type.

5.2.1 Exam Results Overview

The average score of the exam was 61.45% and the median was 61.11%. A t-test was done to determine if there was a significant difference in results between 6th and 7th grade students. The Shapiro-Wilk test gave p-values greater than 0.05 for both the 6th and 7th grade groups, indicating the data is normal. Levene's test for equality of variances gave $p = 0.825$, confirming the variances are not significantly different. This means all assumptions are met and a t-test can be used.

- Null hypothesis H_0: There is no significant difference in the mean exam scores between 6th and 7th grade ($\mu_6 = \mu_7$).

- Alternative hypothesis H_A: There is a significant difference in the mean exam scores between 6th and 7th grade ($\mu_6 \neq \mu_7$).

The t-test gave a high p-value ($t(113) = -0.027$, $p = 0.98$), well above the significance level ($\alpha = 0.05$) which means we fail to reject H_0. Therefore, there is no significant difference in scores between grade levels. In fact, this result indicates that the exam scores between 6th and 7th grade are nearly identical. Figure 5.13 shows a box plot of the test scores between grades.

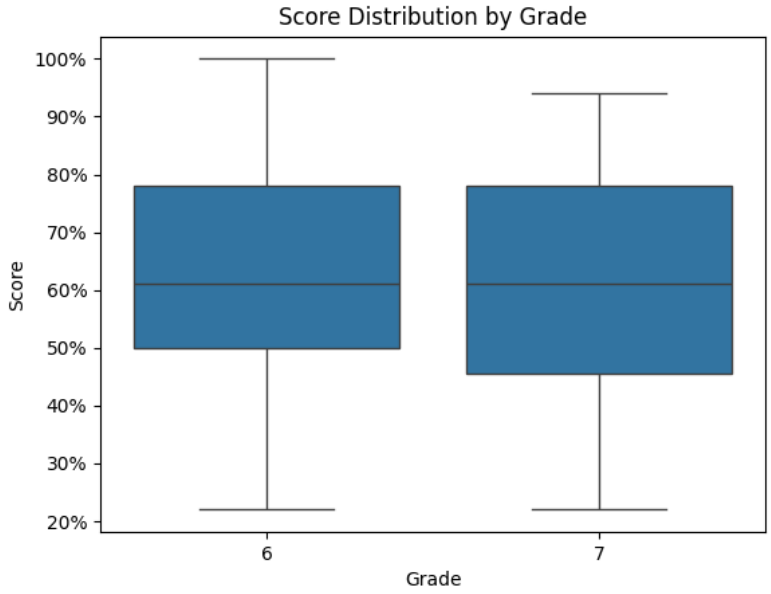


Figure 5.13: Box plots of the score distribution by grade level. The distribution of scores between grade levels is nearly identical.

5.2.2 Tested Misconceptions

The multiple-choice examination contained questions directly targeting variable misconceptions from previous literature. These questions, along with the misconceptions they targeted, can be seen in Table 5.1. Additionally, the overall results of the examination for each specific question targeting a misconception can be seen in Figure 5.14. This figure shows the proportions of students who answered a question correctly, who selected a misconception answer, and who selected a different incorrect answer for every question targeting a misconception.

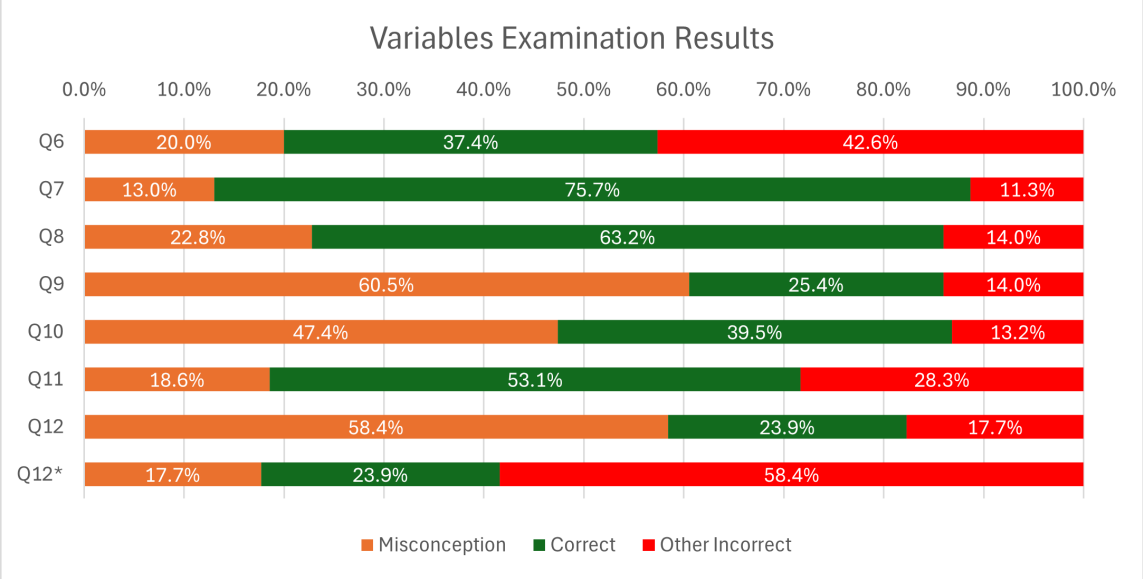


Figure 5.14: A stacked bar showing the proportion of students who selected the misconception answer, correct answer, and other incorrect answers for questions targeting misconceptions.

To compare our results with those of previous studies, the percentage of students who answered the misconception answer in our study and previous studies was used

to conduct a two-tailed two-proportion z-test with a significance level $\alpha = 0.05$. For all tests, $n_1 p_1 \geq 5$, $n_1(1 - p_1) \geq 5$, $n_2 p_2 \geq 5$, and $n_2(1 - p_2) \geq 5$ where p_1 is the percentage of students in our study who chose a misconception answer, and p_2 is the percentage of students in a previous study who chose a misconception answer. This means the assumption for approximate normality was met for all tests, and the two-proportion z-test is appropriate.

For every question that targets a misconception, the following hypotheses were used:

- Null hypothesis H_0: There is no significant difference between the proportion of students who selected a misconception answer in this study and previous studies ($p_1 = p_2$).
- Alternative hypothesis H_A: The proportion of students who selected a misconception answer in this study is significantly different from previous studies ($p_1 \neq p_2$).

If the null hypothesis is rejected and fewer students chose the misconception answer in our study, then it is determined that significantly fewer students formed the misconception. Conversely, if more students in our study chose the misconception answer, it is determined that significantly more students formed the misconception.

Table 5.2 displays the results of the two-proportion z-test of all questions with a misconception. Of all the questions, two showed there was sufficient evidence that significantly fewer students formed the misconception compared to previous studies. For three of the questions, there is sufficient evidence that significantly more students

formed the misconception compared to previous studies. For the remaining questions, there was no significant difference in the proportion of students who formed the misconception compared to previous studies.

5.2.3 New Misconceptions

In addition to the misconceptions explicitly tested for, there were instances where a significant proportion of students chose an incorrect answer in the quiz not targeting a misconception, revealing new misconceptions. For Q5 in the exam, 25% of students incorrectly identified "2024" as an integer, and 45.7% identified 'False' as a boolean despite both being strings (in Python, strings can be written using double or single quotation marks). This suggests many students have trouble recognizing strings when the string represents a different data type in some situations.

Additionally, in Q6 42.6% of students chose an option where the quotation marks are printed when printing a string. The exact example in the quiz is print("Happy Birthday"), and students chose the answer option that included the quotation marks in the output. This shows another misconception students have with string data types and print statements, as they don't understand that the quotation marks are excluded.

The *Variable Data Types* book in CodeKids does address both of these issues, although they are not misconceptions that were specifically targeted. There is text in the book explicitly stating that strings are always surrounded by single or double quotation marks in Python. For the second misconception, where students included

the quotation marks in printed strings, there is also an incorrect answer for one of the questions in one of the activities that includes the quotation marks of a printed string. Despite these examples, it's clear that a significant number of students still formed these misconceptions regarding strings and likely need more practice recognizing when a value is surrounded by quotation marks.

Question	Misconception	Example code and misconception answer from the quiz
Q2	A variable is not stored inside the computer.	A child can think it is stored in the monitor or not stored anywhere.
Q6	Summation on variable assignment.	age = 10 age = 11 print(age) Student says “21” is printed.
Q7	Variables can hold multiple values.	day_of_month = 7 day_of_month = 8 print(day_of_month) Student says “7 8” is printed.
Q8	Variables are assigned expressions.	leap_year = 2024 leap_year = leap_year + 4 print(leap_year) Student says “leap_year + 4” is printed.
Q9	Sequential execution of code.	hokie_score = 0 old_dominion_score = 0 difference = hokie_score + old_dominion_score hokie_score = 37 hokie_score = 17 print(difference) Student says “20” is printed.
Q10	A print statement that contains a string value that happens to be a variable name will print the variable’s value.	soccer_score = 2 soccer_score = 3 print(”soccer_score”) Student says “3” is printed.
Q11	Using the first or previously assigned value of a variable.	food_price = 6 food_price = 5 print(food_price) Student says “6” is printed.
Q12	Natural language semantics of a variable’s name affects the variable’s value.	big_number = 100 small_number = 1 big_number = small_number print(big_number, small_number) Student says “100 1” is printed.
Q12*	Variable assignment works in opposite directions.	big_number = 100 small_number = 1 big_number = small_number print(big_number, small_number) Student says “1 100” is printed.

* This misconception was originally targeted in Q3 but one of the answer choices in Q12 better reflected the misconception from Swidan et al. [15]

Table 5.1: Exam questions and the misconceptions they target.

Question	Our Study	Previous Study	p-value	Result
Q2	4.3%	33.3%[15]	$p = 8.63 \times 10^{-9}$	Reject H_0: fewer students in our study formed the misconception compared to the study from Swidan et al. [15] with 7-17 year old students (mean age 11) using Scratch.
Q7	13%	42.9%[15]	$p = 8.97 \times 10^{-7}$	Reject H_0: fewer students in our study formed the misconception compared to the study from Swidan et al. [15] with 7-17 year old students (mean age 11) using Scratch.
Q6	20%	18.9%[19]	$p = 0.84$	Cannot reject H_0: there is no significant difference in the number of students who formed the misconception in our study compared to the study from Žanko et al. [19] with 10-11 year old students using Python.
Q8	22.8%	25.4%[15] 21.9%[19]	$p = 0.63$ $p = 0.88$	Cannot reject H_0: there is no significant difference in the number of students who formed the misconception in our study compared to the study from Swidan et al. [15] with 7-17 year old students using Scratch (mean age 11) and the study from Žanko et al. [19] with 10-11 year old students using Python.
Q9	60.5%	56.2%[15]	$p = 0.49$	Cannot reject H_0: there is no significant difference in the number of students who formed the misconception in our study compared to the study from Swidan et al. [15] with 7-17 year old students (mean age 11) using Scratch.
Q11	18.6%	18.9%[19]	$p = 0.96$	Cannot reject H_0: there is no significant difference in the number of students who formed the misconception in our study compared to the study from Žanko et al. [19] with 10-11 year old students using Python.
Q10	47.4%	33.68% [19]	$p = 0.04$	Reject H_0: more students in our study formed the misconception compared to the study from Žanko et al. [19] with 10-11 year old students using Python.
Q12	58.4%	20.6%[15]	$p = 3.87 \times 10^{-10}$	Reject H_0: more students in our study formed the misconception compared to the study from Swidan et al. [15] with 7-17 year old students (mean age 11) using Scratch.
Q12*	17.7%	3.9%[15]	$p = 2.30 \times 10^{-4}$	Reject H_0: more students in our study formed the misconception compared to the study from Swidan et al. [15] with 7-17 year old students (mean age 11) using Scratch.

* This misconception was originally targeted in Q3 but one of the answer choices in Q12 better reflected the misconception from Swidan et al. [15]

Table 5.2: Exam questions and their p-values after performing a two-proportion z-test to compare the proportion of students who selected the misconception answer in our study to those in previous studies.

Chapter 6

Discussion

6.1 Survey Discussion

Our first hypothesis was that the storybook format CodeKids uses would be an enjoyable format for the children to learn computational concepts and increase their perceived knowledge of the concept they are learning. Based on the survey results we received, we can confirm this hypothesis. The survey also gave valuable insight into other important aspects, such as the children's prior programming experience and interest in programming. Additionally, we have identified potential areas where the design of the books and tools used in the books could be improved.

6.1.1 Lack of Prior Programming Experience

The majority of students reported minimal to no prior programming experience, despite the *Computer Science Standards of Learning for Virginia Public Schools* indicating that students should be using block-based programming environments in the second grade [16]. This lack of experience is shown in Figure 5.1 and Figure 5.2 as most students ranked themselves as having minimal to no experience, and the

majority stated they have never used a programming language before. Given that the Virginia state learning standards recommend students be introduced to block-based programming in the second grade [16], this finding is noteworthy since the students in our study are in the 6th and 7th grades. Additionally, the finding that 74% of students have never used variables - depicted in Figure 5.4 - is contradictory to the learning standards, which state students should learn variables by the fourth grade.

Another notable observation is that older students are not more likely to have experience programming than younger students. The Spearman correlation test between the different age groups did not show a significant correlation in their prior programming experience ($\rho = -0.029,\ p > 0.05$) or prior use of variables ($\rho = 0.188,\ p > 0.05$), suggesting age alone is not a determining factor in prior programming experience. Figure 5.11 and the "Prior Programming Experience by Age Group" chart in Figure 5.12 visualize the similar proportions between age groups. This contrasts with the theoretical expectation that older students would have more experience programming if schools were meeting the learning standards.

Despite this lack of experience, most students are at least somewhat interested in learning how to program, as depicted in Figure 5.3. Not only does this show that current teaching practices are falling behind the learning standards, but they are also failing to meet the students' interest in computer science education. Systemic factors such as unequal access to resources, teacher training gaps, and curricular inconsistencies among schools could contribute to this. It also emphasizes the need to prioritize computational thinking earlier in children's education. Doing this will

not only help schools meet the educational standards but also help children develop the necessary thinking skills in today's technology-driven world.

While addressing these systemic factors is theoretical and is out of the scope of this paper, it does highlight the need for equitable access to programming resources, teacher training, and age-appropriate curricula. Future work could investigate the reasons why few students have significant programming experience, and CodeKids and other educational resources could be further adapted to address any of these factors that may be preventing students from receiving a computer science education in elementary and middle school.

6.1.2 Use of Program Visualizations

Program visualizations, such as Python Tutor, allow students to visualize program execution and are a form of multimedia learning. These tools can be used in education to help students understand complex concepts, such as variable manipulation. It was hypothesized that integrating Python Tutor in the books would help children better understand variables and dispel the common misconceptions associated with variables, especially ones related to their manipulation and the sequential execution of code. However, the students' perception of Python Tutor is mixed, as shown in Figure 5.5. This suggests that Python Tutor in its current form is not useful for many students to learn about variables, though more study is needed to measure learning gains.

Several factors can play into this mixed perception. For one, the implementation is

limited by technical constraints. Because Python Tutor can only be embedded using iframes, it is difficult to modify or interact with Python Tutor's state from the main website. For example, we cannot stop the students from skipping Python Tutor and going to the next page before reaching the end of the program because we don't know where the user is in the visualization. We also cannot ask questions at specific points of the code because we don't know when the student hits the "Next" button or which line of code they are on. These limitations reduce the tool's effectiveness as an interactive learning aid in CodeKids.

Overall, the current implementation of Python Tutor as a program visualization in CodeKids does not seem useful for many students, despite the advantages that program visualizations can have in computer science education. Future work should explore ways to make program visualizations more interactive, such as embedding questions or prompts at key points in the code's execution. Doing so will likely be more engaging for students since it will allow them to respond to the material instead of passively absorbing it [5]. Additionally, further studies should be done that directly investigate the usage of program visualization tools on young learners and the best practices for designing these tools. These efforts can positively impact and transform the teaching practices and learning outcomes for young students in computer science.

6.1.3 Perceptions of Books

Students generally expressed positive perceptions of the books and their activities. As shown in Figure 5.7 and Figure 5.6, the majority of students found the books' examples and activities enjoyable, and a plurality of students rated the books as somewhat difficult. This indicates a balanced difficulty that is not overly challenging or too simplistic. However, a notable number of students perceived the books as "easy", and a smaller group rated them as "very difficult". This suggests that while the books were an appropriate difficulty level for many students, there is room to accommodate students at both ends of the knowledge spectrum.

Statistical analysis showed there is no correlation between the age of the students and how difficult they perceived the books ($\rho = -0.097,\ p = 0.345$) or their enjoyment of the examples and activities ($\rho = -0.162,\ p = 0.115$). This can be seen from the "Difficulty of Books by Age Group" and "Liked Examples and Activities by Age Group" charts in Figure 5.12. This indicates the books' difficulty was perceived similarly across the age ranges of 11-14 years old, and the books' design, including content, pacing, examples, and activities, catered equally well to the students across this age group. This finding demonstrates that the books are broadly applicable within this age group without requiring major age-specific adaptations.

These findings highlight the books' overall success in engaging the students while also showing opportunities for improvement. Future work could look to incorporate adaptive elements to address the learning needs of particular students. For example, questions could be generated or chosen for specific students based on how well they

have answered questions in the past. This would allow for more difficult questions to be created for students who show more knowledge and simpler questions for students with less knowledge. Additionally, it could be worthwhile to explore further how different age groups perceive the difficulty and enjoyment of the books. This could be done by letting younger elementary school students and older high school students use the books to see how perceptions of difficulty and enjoyment shift across a broader age range. Such studies could provide deeper insights into how to effectively design educational materials for specific age groups.

6.1.4 Perceived Knowledge and Self-Efficacy Regarding Variables

It was hypothesized that the CodeKids books would increase the students' perceived self-knowledge towards variables. Based on our statistical analysis of how the students ranked their knowledge of variables before and after using the CodeKids books, we can confirm this hypothesis ($W = 262,\ p < 0.001$).

The results indicate a notable increase in perceived knowledge with the majority of students ranking their perceived knowledge by at least one more rank, which can be seen in Figure 5.10. This suggests the books were effective in helping students conceptualize variables in a meaningful way and boost their confidence in understanding the topic.

However, while perceived knowledge did increase among students, it's important to acknowledge that perceived knowledge does not always reflect actual knowledge.

Overconfidence can lead students to believe they understand a concept more than they do. This was a known limitation when creating the survey and highlights the need for additional assessments to measure actual learning outcomes of the students. Based on the exam results and the comparison of misconceptions rates, it is evident that students still have misunderstandings, particularly regarding the manipulation of variables, and a long-term study will need to be completed to determine the learning outcomes of variables compared to the original instructional material. Future work should combine the self-reported knowledge with objective measures of knowledge to give a more comprehensive understanding of the books' impact on student knowledge of computational concepts.

Despite this limitation, perceived knowledge can still be valuable in learning. An increase in perceived knowledge can help increase the motivation and engagement of students and encourage further learning of the material [14]. Students may also be more confident in pursuing future challenges, particularly when they begin learning about conditionals and loops. Overall, the increase in perceived knowledge could increase students' self-efficacy, which is often correlated with better learning outcomes [18].

6.2 Examination Discussion

In our second research question, we aim to compare the misconception rates among students using these digital books to the rates of prior research studies and determine possible factors that could contribute to observed differences (or lack thereof). As can

be seen in Table 5.2, for many of the questions, there wasn't a significant difference. However, there were still three questions where significantly more students formed the misconception and two where significantly fewer formed the misconception. Of the two questions where significantly fewer students formed the misconception, both had to do with the general definition of variables as storage spaces. This suggests that the books were good at conceptualizing the general concept of variables for children, but they still struggle with their manipulation in code. This observation is consistent with findings from Luo et al. [9], who observed with 4th grade students that students recognized variables as storage spaces but struggled to operate on them. In contrast, of the three questions where significantly more students formed the misconception, all three were related to how variables can be manipulated in code, including using the previously assigned value, the natural language semantics of the variable affecting its value, and variable assignment working in opposite directions.

While findings suggest the books are good for conceptual knowledge, there is no immediate evidence to suggest they helped prevent the formation of misconceptions. A longer-term study is necessary to assess whether the books affect misconception formation rates compared to the teacher's current instructional material.

6.2.1 Possible Factors that Contributed to the Formation of Misconceptions

There are several factors and limitations identified that could potentially explain the results obtained. Particularly for those misconceptions where there is no significant

difference or significantly more children who formed the misconception.

Environment of Previous Studies

The study by Swidan et al. [15] wasn't conducted in a traditional classroom setting. Instead, it was done at a science museum where children were asked to join the experiment. Additionally, they filtered out students who did not have prior programming experience or who appeared to be guessing answers. This means all of the students in their study had previous experience in programming, meanwhile the majority of our students had little to no experience. We also don't know which students could have been guessing during the examination, so filtering them out is impossible. Lastly, their study had a significant number of students who were 14 or older, while the majority of our students were 11-13 years old.

The study by Žanko et al. [19] was done in a traditional classroom setting similar to ours. They also stated that the students in their study had no prior programming experience. However, their in-class study time was four weeks while our time was two weeks. This difference in time could have given the students in their study more time to learn the material and perform better than our students. However, their students were slightly younger than ours, with all of their students being 10-11 years old.

As noted before, in the future a more robust study will need to be conducted to determine if the books lessen the proportion of students who form misconceptions. These discrepancies with previous studies highlight this need.

Python Tutor Implementation

The limitations of Python Tutor in CodeKids were discussed in Subsection 6.1.2. These limitations could have negatively impacted Python Tutor's capability to prevent the formation of misconceptions. As noted before, a new tool should be made that addresses these limitations and allows the integration of questions during the program's execution. Doing so will give the opportunity for students to respond to the program visualization instead of passively absorbing the information, and it is believed that having more interaction with a program visualization increases learning outcomes [5]. Research should also be done on the effects of program visualization tools on young learners to inform design choices when creating such tools. Doing so could not only prevent the formation of misconceptions for advanced topics like variables, but also transform the teaching practices currently used to teach K-8 students computer science.

Impact of Pass/Fail Grading

The teacher mentioned they are not able to grade student assignments on a traditional A-F scale. Due to the school's policy, they can only give students a passing or failing grade, which is only dependent on completion of the exam and not the student's score. While the pass/fail grading scheme could have potential learning benefits, this grading practice could still have negatively affected this study since students may not have felt motivated to try and do well on the test. Butcher et al. [2] noted that instructors felt their students' motivation and effort were lower in

their first-semester pass/fail classes and that letter grades of first-semester students declined 0.13 grade points compared to first-semester students in previous years (the 0.13 decline was calculated based on the letter grade the first-semester students would have received if the class wasn't pass/fail). Although this study was done for college-level students, these types of negative effects could still cascade down to younger students' motivation and effort and result in lower exam performance than if a traditional A-F grading scale were used.

Unfortunately, this isn't an issue we can remedy currently since this decision was made by the schools, and determining if the pass/fail grading scheme affects exam scores is out of the scope of this study. In future studies, we could look for schools that follow a traditional grading scheme to ensure this variable isn't a contributing factor in the formation of misconceptions.

No Post-Book Assessment

Currently, the only way for students to test their knowledge of the topic they are learning is with the per-page questions they answer in the books. These questions give explanations for wrong answers, but it is still possible for the students to click answers until they get the correct one and brute force their way through the books without meaningfully connecting with the content. Although we did not collect interaction traces to confirm this, since students were not required to create accounts for this study, this behavior is still a concern, as such behavior could lessen the effect of preventing the misconceptions that the books target.

While the post-intervention examination offers a summative assessment of student understanding, it does not provide real-time feedback during the learning process. To address this, we could integrate brief post-book quizzes for students to take at the end of each book that assess students' conceptual understanding. These quizzes could offer personalized review suggestions based on a student's performance and encourage them to revisit concepts if necessary. Additionally, future books could be locked until the student has the necessary prerequisite knowledge to continue with more concepts. This system could also give teachers insights into the topics and misconceptions their students are struggling with the most by providing them with analytics of the questions students get wrong most frequently. Overall, this approach would enhance CodeKids' role as a learning tool and formative assessment resource for educators.

6.2.2 Correlation Between Score and Grade Level

Statistical analysis of the student exam scores and their grade levels showed no significant correlation. Students from 6th and 7th grade performed similarly, indicating grade level was not a determining factor when learning about variables. This finding is notable, as it suggests older students did not demonstrate a measurable advantage over younger students when learning a complex computational concept such as variables. It also means students are well behind the Virginia computer science learning standards and further supports findings discussed in Subsection 6.1.1. By sixth grade, students should be able to write programs with nested conditional control

structures, which is well beyond the understanding of variables [16].

This also suggests that conceptual understanding of programming concepts, such as variables, may be more dependent on previous exposure to the topic than the age and development of the students. This further emphasizes the importance of exposing children to computational concepts at a younger age. Future work could look more into comparing students' understanding of computational concepts of those who have had previous exposure to those who haven't. Additionally, future studies could investigate whether different forms of assessment, such as coding exercises and multiple-choice quizzes, reveal score differences across grade levels.

6.3 Teacher Perception of the Books

As mentioned in Sub-section 3.1.3, a K-8 educator reviewed each of the books and provided feedback on their design. Overall, the reception of the books was positive and highlighted their accessibility, pedagogical clarity, and student engagement. The teacher praised the "Read to Me" feature of the books, which allows students to click on a sentence in the book and have it be read to them, and is meant to make the books more accessible. It was noted by the teacher that many digital learning tools don't include audio support, which can be disengaging for students who struggle to read. This made the feature in CodeKids encouraging for the teacher.

The teacher also appreciated the feedback given to students for incorrect answers, and cited it as being great for guiding students to the correct answer. They also

mentioned that the feedback given is detailed and educational, offering students a pathway for understanding rather than a simple correction. Furthermore, the teacher considered the vocabulary and explanations of the books to be age-appropriate and clear, supporting students' conceptual understanding of computer science terms like "variables", "logical operators", and "flow of execution".

The design and interactivity of the books were also praised, with the teacher stating the visuals were bright and bold, and the exercises were intuitive and engaging. In the *Life of Moose* book, the teacher stated students will love to learn about Moose and that it is a "great way to engage students upon opening this book". Lastly, the teacher noted how the books aligned well with the *Computer Science Standards of Learning for Virginia Public Schools* [16].

While the teacher gave some suggestions for simplifying terminology and navigation between books, the overall sentiment was that the CodeKids platform was an engaging way for students to learn computer science. However, these are the opinions of a single teacher, and more K-8 educators should review the books to get a more diverse outlook on them.

6.4 Limitations of Study

While these digital books were designed to prevent the formation of misconceptions for children learning computer science, we can not conclusively determine if these books had any effect on the rates of misconception relative to their current instruc-

tional material. Currently, we can only compare our results with those of previous studies. Future long-term studies can better analyze the books and their effect on student learning outcomes. Doing so would mean having a control and experimental group to determine if the books help prevent students from forming misconceptions compared to the material they currently use to learn computer science. Conducting a longer-term study would also allow us to use more of the books in Chapter 4, including the conditional and loops books.

Additionally, potential biases are present in the study. For one, convenience sampling bias is potentially present since the books were used in a single middle school with a specific STEM teacher, which is not representative of broader student populations. This study also assumes students had minimal to no prior programming experience. While the survey confirms most do not, students with prior experience may have had an advantage in understanding the material and skew perceptions of difficulty and learning outcomes. Lastly, survey response bias could have contributed to students overestimating or underestimating their programming experience, interest, and perceived knowledge. Students could have also answered in ways they thought were expected rather than reflecting their true opinions.

Chapter 7

Conclusion

In this study, we introduced a website called CodeKids to teach children computer science concepts using an interactive digital storybook format. While the platform hosts a variety of books covering multiple computer science concepts, this study focuses on several books that were created to target common misconceptions K-8 students form when learning the computational concepts of variables, conditions, and loops. We aimed to observe how the students interact with books and if the students enjoyed learning from these books. We also wanted to compare the misconception rates of students using our books to the rates of previous studies.

7.1 Key Findings

In terms of how the children perceived the books, we can conclude from our survey that the students found the CodeKids platform to be an enjoyable medium to learn computer science from, with 82.5% of students indicating they enjoyed the activities and examples in the books and most students believing the books were not too difficult. A significant number of students also believed their knowledge of variables

increased after using the books, which could help motivate students to learn more about computer science. Still, there are a significant number of students who believe the books were too difficult or too easy, and it's important to design ways to accommodate these students.

The survey results also gave other valuable insights. For one, the vast majority of students had no or little prior experience in programming, and about three-quarters of students have never used variables before, despite the Virginia learning standards recommending students use variables by the fourth grade. This highlights the need to ensure schools have the resources to meet the learning standards and expose children to computer science at an earlier age. Additionally, the age of the students played no determining factor for any aspects of the children's perception of the books or programming experience. This suggests the books were designed well for this particular age group.

As for the examination, age also didn't play a determining factor. The grades of students between 6th and 7th grade were virtually identical. The exam also didn't suggest that children may be less likely to form the common computer science misconceptions that the books were created to target. For most of the variable misconceptions targeted, there was either no difference in the number of students that formed the misconception or significantly more students formed the misconception compared to other studies. Several potential factors were identified that could contribute to these results, including the environment of other studies, Python Tutor limitations, and the classroom grading scheme. Ultimately, though, a longer-term research study will need to be conducted to determine if the books help prevent the

formation of the misconception that compares the books to the current materials they use to learn computer science.

7.2 Contributions to the Field

This study contributes to the growing body of research into K-12 computer science education. It introduces a novel way to teach computer science to children through a digital storybook medium and demonstrates the feasibility of creating this medium. This format proved to be an enjoyable format for children to learn computer science, however, misconceptions rates remained largely persistent and consistent from previous studies. It underscores the challenges of addressing misconceptions still faced by young students and the importance of designing tools that actively engage students in the learning process. Such tools could include program visualizations that allow for more interaction by students by answering questions during their execution.

This study also further exposes the discrepancies between the learning standards set out for schools and students' actual knowledge. While the specific reasons for this have not been identified by this study, it does highlight the gap between expected learning outcomes and students' understanding. Considering many students expressed interest in programming, this gap may be hindering their progress. This contributes to the ongoing discussion on how to properly incorporate computer science education into K-8 education that is accessible and aligns with educational standards.

While the CodeKids platform shows promise as an engaging and accessible tool for teaching computer science to children, further research is needed to optimize its effectiveness in reducing misconceptions and supporting diverse learners. Overall, this study helped lay down the foundation for future efforts to design and evaluate digital learning tools that help K-8 students succeed in computer science education.

7.3 Future Work

Future work should conduct a long-term study comparing CodeKids to the traditional instructional material used by the school, using control and experimental groups. Doing so, we can compare how effective the CodeKids books are in teaching computer science to students compared to the instructional material they currently use. Because elementary and middle school teachers typically retain the same students throughout the year, classrooms (instead of semesters) would likely serve as the units of comparison. Ideally, multiple classrooms at the same grade level across different schools or within a large school could be randomly assigned to either the control or experimental groups. This study would help us more definitively compare student outcomes, such as conceptual understanding and misconception rates, and also offer stronger evidence of the educational impact of CodeKids.

The perceptions of more K-8 educators on the books should also be worked on in the future. Currently, the books have been reviewed by only a single teacher. A survey should be given to more K-8 educators to assess aspects such as clarity of explanations, engagement, accessibility, and perceived impact on student learning.

Doing so will give us more diverse opinions on the books we can use to further enhance the design of the CodeKids platform.

Additionally, future research should be done on how to design program visualization tools that are more interactive and adaptive than Python Tutor. Python Tutor's limitations and mixed reception among students support this need. Program visualizations have the potential to transform how computer science education is done in K-8 classrooms, and understanding how to best design them for young learners is critical. A good first step would be to ask questions at specific points in the code, which would allow students to actively engage with the visualization. Not only could this help computer science education overall, but also help reduce the common misconceptions.

Bibliography

[1] Philip Sheridan Buffum, Kimberly Michelle Ying, Xiaoxi Zheng, Kristy Elizabeth Boyer, Eric N. Wiebe, Bradford W. Mott, David C. Blackburn, and James C. Lester. Introducing the computer science concept of variables in middle school science classrooms. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 906–911, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351034. doi: 10.1145/3159450.3159545. URL https://doi.org/10.1145/3159450.3159545.

[2] Kristin Butcher, Patrick J. McEwan, and Akila Weerapana. Making the (letter) grade: The incentive effects of mandatory pass/fail courses. *Education Finance and Policy*, 19(3):385–408, 07 2024. ISSN 1557-3060. doi: 10.1162/edfp_a_00401. URL https://doi.org/10.1162/edfp_a_00401.

[3] Joseph M. Furner. Using children's literature to teach mathematics: An effective vehicle in a stem world. *European Journal of STEM Education*, 3(3), Sep 2018. doi: 10.20897/ejsteme/3874.

[4] Marcos J. Gomez, Marco Moresi, and Luciana Benotti. Text-based programming in elementary school: A comparative study of programming abilities in children with and without block-based experience. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Educa-*

tion, ITiCSE '19, page 402–408, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368957. doi: 10.1145/3304221.3319734. URL https://doi.org/10.1145/3304221.3319734.

[5] Scott Grissom, Myles F. McNally, and Tom Naps. Algorithm visualization in cs education: comparing levels of student engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, page 87–94, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136420. doi: 10.1145/774833.774846. URL https://doi.org/10.1145/774833.774846.

[6] Shuchi Grover and Satabdi Basu. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, page 267–272, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346986. doi: 10.1145/3017680.3017723. URL https://doi.org/10.1145/3017680.3017723.

[7] Figma Inc. Figma. URL https://www.figma.com.

[8] Sang Joon Lee, Gregory M. Francom, and Jeremiah Nuatomue. Computer science education and k-12 students' computational thinking: A systematic review. *International Journal of Educational Research*, 114:102008, 2022. ISSN 0883-0355. doi: https://doi.org/10.1016/j.ijer.2022.102008. URL https://www.sciencedirect.com/science/article/pii/S0883035522000866.

[9] Feiya Luo, Wei Yan, Ruohan Liu, and Maya Israel. Elementary students' understanding of variables in computational thinking-integrated instruction: A

mixed methods study. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, SIGCSE 2022, page 523–529, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390705. doi: 10.1145/3478431.3499323. URL https://doi.org/10.1145/3478431.3499323.

[10] Monica M. McGill and Laycee Thigpen. Extrinsic barriers to integrating computer science in elementary school subject areas in the united states. In *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research*, WiPSCE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400710056. doi: 10.1145/3677619.3678116. URL https://doi.org/10.1145/3677619.3678116.

[11] Alexandros Merkouris and Konstantinos Chorianopoulos. Introducing computer programming to children through robotic and wearable devices. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE '15, page 69–72, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450337533. doi: 10.1145/2818314.2818342. URL https://doi.org/10.1145/2818314.2818342.

[12] Daisuke Saito, Hironori Washizaki, Yoshiaki Fukazawa, Tetusya Yoshida, Isumu Kaneko, and Hirotaka Kamo. Learning effects in programming learning using python and raspberry pi: Case study with elementary school students. In *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*, pages 1–8, 2019. doi: 10.1109/TALE48000.2019.9225866.

[13] T. Sapounidis, Stavros Demetriadis, and Ioannis Stamelos. Evaluating children performance with graphical and tangible robot programming tools. *Personal and Ubiquitous Computing*, 05 2014. doi: 10.1007/s00779-014-0774-3.

[14] Dale H. Schunk and Frank Pajares. Chapter 1 - the development of academic self-efficacy. In Allan Wigfield and Jacquelynne S. Eccles, editors, *Development of Achievement Motivation*, Educational Psychology, pages 15–31. Academic Press, San Diego, 2002. doi: https://doi.org/10.1016/B978-012750053-9/50003-6. URL https://www.sciencedirect.com/science/article/pii/B9780127500539500036.

[15] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. Programming misconceptions for school students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, page 151–159, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356282. doi: 10.1145/3230977.3230995. URL https://doi.org/10.1145/3230977.3230995.

[16] Virginia Department of Education. Computer science standards of learning for virginia public schools, 2024. URL https://www.doe.virginia.gov/home/showpublisheddocument/57226/638612962892700000.

[17] Danli Wang, Lan Zhang, Chao Xu, Haichen Hu, and Yunfeng Qi. A tangible embedded programming system to convey event-handling concept. In *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '16, page 133–140, New York, NY, USA, 2016.

Association for Computing Machinery. ISBN 9781450335829. doi: 10.1145/2839462.2839491. URL https://doi.org/10.1145/2839462.2839491.

[18] Barry J. Zimmerman. Self-efficacy: An essential motive to learn. *Contemporary Educational Psychology*, 25(1):82–91, 2000. ISSN 0361-476X. doi: https://doi.org/10.1006/ceps.1999.1016. URL https://www.sciencedirect.com/science/article/pii/S0361476X99910160.

[19] Žana Žanko, Monika Mladenović, and Divna Krpan. Analysis of school students' misconceptions about basic programming concepts. *Journal of Computer Assisted Learning*, 38(3):719–730, 2022. doi: https://doi.org/10.1111/jcal.12643. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/jcal.12643.

Appendices

Appendix A

Variables Examination

Q. 1 What is a variable in programming?

- ○ a. A number that stays the same throughout a program.
- ○ b. A part of the computer that changes how fast it can run.
- ● c. A place to store data in a program.
- ○ d. A command that forces the computer to display a message.

Q. 2 Where is a variable stored? A variable is not stored inside the computer.

- ● a. Computer's memory
- ○ b. Computer's monitor
- ○ c. Computer's keyboard
- ○ d. It is not stored anywhere

Q. 3 Which of these is the wrong way to assign a variable named *state*? Variable assignment works in opposite direction.

- ○ a. state = 6
- ○ b. state = "Virginia"
- ● c. "Virginia" = state
- ○ d. state = True

Q. 4 What do each of these data types represent? Choose one option per row.

	A Sequence of letters.	Any whole number.	A True or False value
Integer	○	●	○
String	●	○	○
Boolean	○	○	●

Q. 5 Match the data with its data type. Choose one option per row.

	Integer	String	Boolean
2024	●	○	○
“2024”	○	●	○
True	○	○	●
“Hello”	○	●	○
‘False’	○	●	○

Q. 6 What is printed by this code? Summation on variable reassignment.

```
1  age = 10
2  print("Happy birthday!")
3  age = 11
4  print(age)
```

○
```
Happy birthday!
21
```

●
```
Happy birthday!
11
```

○
```
"Happy birthday!"
11
```

Q. 7 What is printed by this code? Variables can hold multiple values.

```
1    day_of_month = 7
2    print("It's the next day!")
3    day_of_month = 8
4    print(day_of_month)
```

○
```
It's the next day!
```

●
```
It's the next day!
8
```

○
```
It's the next day!
7 8
```

Q. 8 What is printed by this code? Variables are assigned expressions.

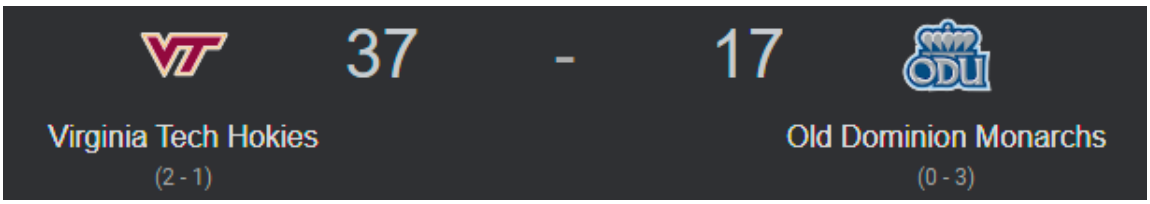
```
1    leap_year = 2024
2    print("When is the next leap year?")
3    leap_year = leap_year + 4
4    print(leap_year)
```

○
```
When is the next leap year?
leap_year + 4
```

○
```
When is the next leap year?
2024
```

●
```
When is the next leap year?
2028
```

Q. 9 What is printed by this program? Sequentiality of code.




```
1    hokie_score = 0
2    old_dominion_score = 0
3    difference = hokie_score - old_dominion_score
4    print("The game has finished!")
5    hokie_score = 37
6    old_dominion_score = 17
7    print(difference)
```

- ●
  ```
  The game has finished!
  0
  ```
- ○
  ```
  The game has finished!
  20
  ```
- ○
  ```
  The game has finished!
  54
  ```

Q. 10 What is printed by this program? Print statement contains a string value that happens to be a variable name.

```
1    soccer_score = 2
2    print("Goal!!!")
3    socer_score = 3
4    print("soccer_score")
```

○
```
Goal!!!
3
```

●
```
Goal!!!
soccer_score
```

○
```
Goal!!!
"3"
```

Q. 11 What is printed by this program? Using the first or previously assigned value of a variable.

```
1    food_price = 6
2    print("$1 off!")
3    food_price = 5
4    print(food_price)
```

○
```
$1 off!
6
```

○
```
$1 off!
food_price
```

●
```
$1 off!
5
```

Q. 12 What is printed by this program? Natual language semantics of a variable affects the variable's name.

```
1    big_number = 100
2    small_number = 1
3    big_number = small_number
4    print("Big number: " + str(big_number))
5    print("Small number: " + str(small_number))
```

○
```
Big number: 100
Small number: 1
```

○
```
Big number: 1
Small number: 100
```

●
```
Big number: 1
Small number: 1
```

Appendix B

Survey

What is your age? *

Your answer

How would you rate your previous experience with programming? *

	1	2	3	4	5	
No Experience 😐	○	○	○	○	○	Very Experienced 😜

Which, if any, of these programming languages have you used before? *

- [ ] Scratch
- [ ] Blockly
- [ ] Python
- [ ] Java
- [ ] None
- [ ] Other:

Are you interested in programming? *

	1	2	3	4	5	
No 😩	○	○	○	○	○	Yes! 😜

Have you used variables in programming before reading these books? *

- ◯ Yes
- ◯ No

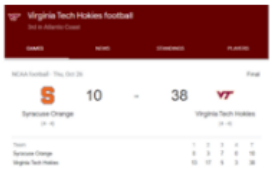
How useful was Python Tutor to learn about variables? *

	1	2	3	4	5	
Not Useful 😟	◯	◯	◯	◯	◯	Very Useful 😀

How difficult were the books? *

	1	2	3	4	5	
Easy! 😀	◯	◯	◯	◯	◯	Very difficult 😬

Did you like the examples/activities the books used? *



We always read the code from top to bottom. The line numbers have been provided. (Yellow column).

```
1  syracuse_score = 10
2  hokies_score = 38
3  different = hokies_score - syracuse_score
4  hokies_score = 23
5  syracuse_score = 22
6  print(different)
```

```
therapy_dog_left = "Derek"
therapy_dog_middle = 'Epcot'
therapy_dog_right = "Josie"
```

- ◯ Yes! 😜
- ◯ No. 😟

If you didn't like the activities, can you please explain why?

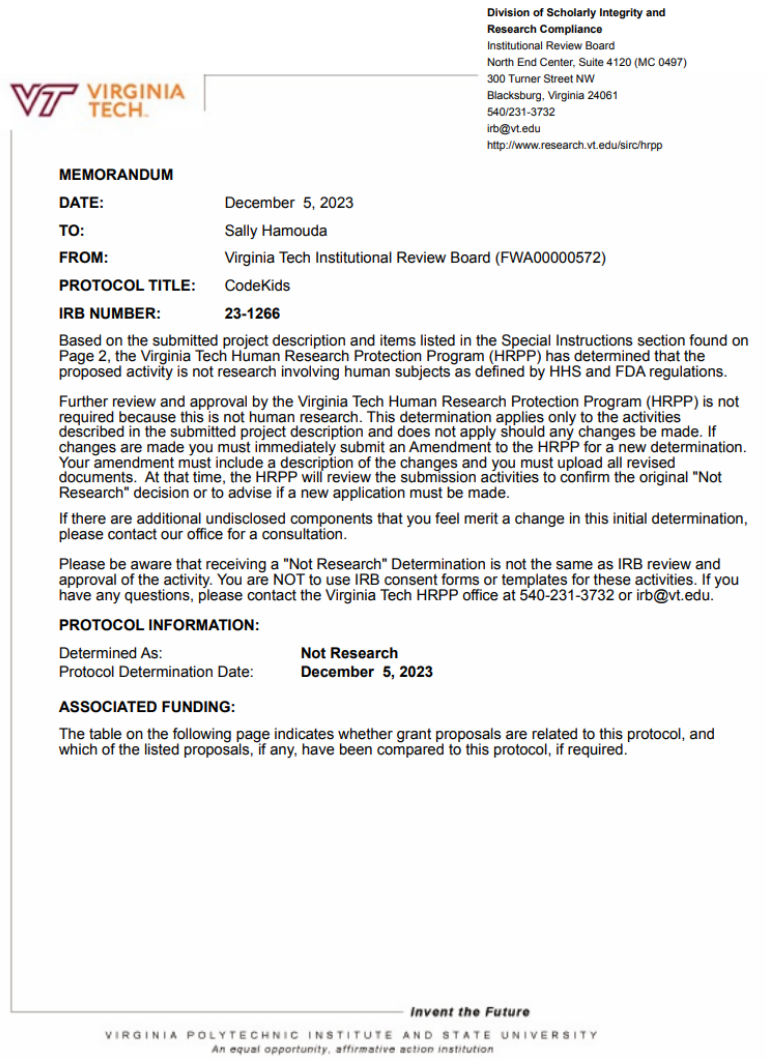
Your answer

How would you rate your knowledge of variables before and after reading the books? *

	No knowledge 😐	Little knowledge 🙄	Some knowledge 😊	A lot of knowledge 😜
Knowledge of variables before reading the books.	◯	◯	◯	◯
Knowledge of variables after reading the books.	◯	◯	◯	◯

Appendix C

IRB Approval Letter



VIRGINIA TECH

Division of Scholarly Integrity and Research Compliance
Institutional Review Board
North End Center, Suite 4120 (MC 0497)
300 Turner Street NW
Blacksburg, Virginia 24061
540/231-3732
irb@vt.edu
http://www.research.vt.edu/sirc/hrpp

MEMORANDUM

DATE: December 5, 2023

TO: Sally Hamouda

FROM: Virginia Tech Institutional Review Board (FWA00000572)

PROTOCOL TITLE: CodeKids

IRB NUMBER: **23-1266**

Based on the submitted project description and items listed in the Special Instructions section found on Page 2, the Virginia Tech Human Research Protection Program (HRPP) has determined that the proposed activity is not research involving human subjects as defined by HHS and FDA regulations.

Further review and approval by the Virginia Tech Human Research Protection Program (HRPP) is not required because this is not human research. This determination applies only to the activities described in the submitted project description and does not apply should any changes be made. If changes are made you must immediately submit an Amendment to the HRPP for a new determination. Your amendment must include a description of the changes and you must upload all revised documents. At that time, the HRPP will review the submission activities to confirm the original "Not Research" decision or to advise if a new application must be made.

If there are additional undisclosed components that you feel merit a change in this initial determination, please contact our office for a consultation.

Please be aware that receiving a "Not Research" Determination is not the same as IRB review and approval of the activity. You are NOT to use IRB consent forms or templates for these activities. If you have any questions, please contact the Virginia Tech HRPP office at 540-231-3732 or irb@vt.edu.

PROTOCOL INFORMATION:

Determined As: **Not Research**
Protocol Determination Date: **December 5, 2023**

ASSOCIATED FUNDING:

The table on the following page indicates whether grant proposals are related to this protocol, and which of the listed proposals, if any, have been compared to this protocol, if required.

SPECIAL INSTRUCTIONS:

This activity does not meet the federal definition of research. The purpose of this study is to help Floyd Elementary school teachers introduce coding to kids through Codekids Website and gain feedback from the students and the teachers to enhance the performance of the tool. The results will be used to develop the program further and will not be generalizable, therefore this is not considered research.

***Please note, this determination only covers the activities described in this determination submission. Any future plans to further analyze or compare will need to be submitted and approved by the HRPP office before analyses or comparisons can occur.

Date*	OSP Number	Sponsor	Grant Comparison Conducted?

* Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this protocol is to cover any other grant proposals, please contact the HRPP office (irb@vt.edu) immediately.